SYSTEM BANK

SMC#1

アスキー・システム・バンク(SMC#1)

SMC-777ガイドブック

更科潤二 & ANIMAL HOUSE著

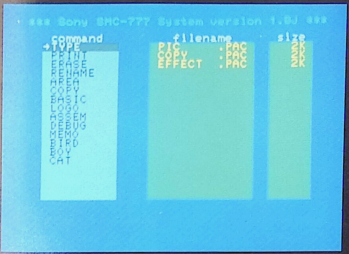
アスキー出版局

SMC-777 ガイドブック

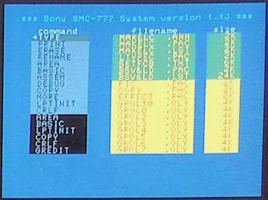
更科潤二 & ANIMAL HOUSE著

Sony Filer

SMC-777に搭載されたオペレーティング・システム，Sony Filer は，誰にもわかり易いメニュー形式．やりたいことを，簡単に選べます



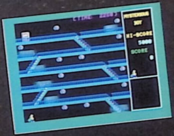
▲オリジナルディスクによるメニュー画面では，カーソルパッドを使って好きなコマンドを簡単に選べます



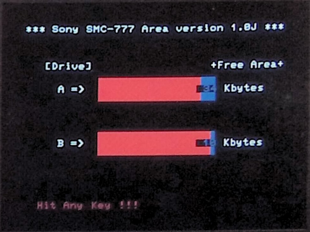
▲増設ドライブを接続したときのメニュー画面です．Aドライブの内容とBドライブの内容は，別の色を付けて区別します

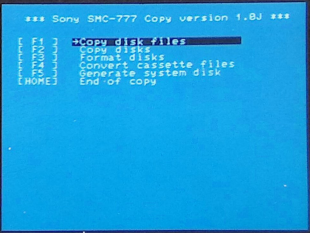
▼AREAコマンドでは，ディスクの使用量をビジュアルに表示します

◀GAMES



GAMES▶

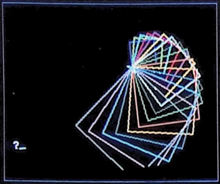




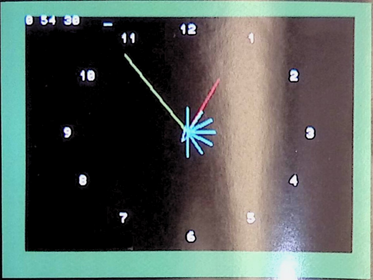
▶COPYコマンドのメニュー画面．ファンクションキーで仕事を選び，ディスクのフォーマッティングやコピー等を行います

Logo Graphics

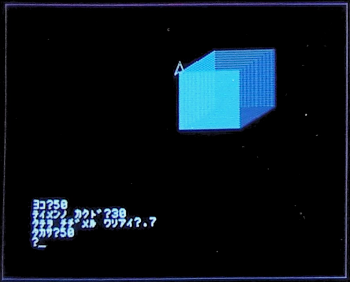
SMC-777は，タートル・グラフィックスで有名な言語，Logoを標準搭載．タートルに命令することで，いろいろな絵を描くことができます



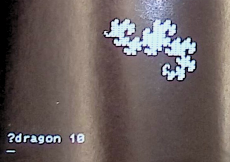
▲タートル・グラフィックスの例としてよく出てくる模様．SMC-777のカラー機能を使えば，カラフルに描くこともできます



▲タートル・グラフィックスのちょっと変った使い方の例として，時計を作ってみました．アラーム機能付きです（リスト3-11，3-12）



▲タートル・グラフィックスを使って描いた立体柱．平行四辺形を沢山つみ上げれば，立派なハシラができあがります（リスト3-3，3-4）

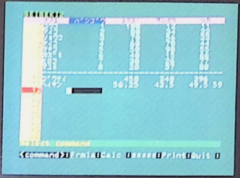


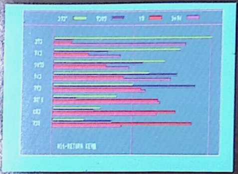
◀Logoの再帰呼出し（リカーシブコール）を利用してドラゴン曲線を描いてみました．10次と指定すると，こんな形です（リスト3-14）

▶こちらは11次のドラゴン曲線です（リスト3-14）

BASIC & Graphics

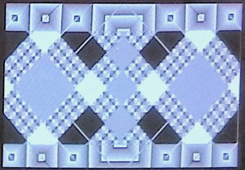
SMC-777はグラフィックス機能の充実したパソコン．BASICを使って様々な絵を描くことができます．特にオプションのカラーパレットを装着すると，4096色のなかから好みの中間色を選ぶこともできます

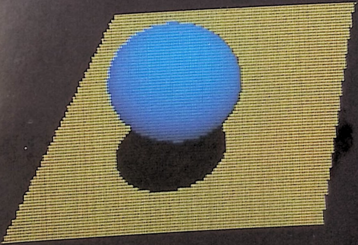


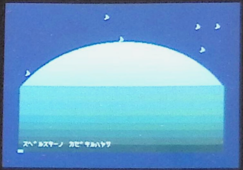


▲パレットを利用して，球を描いています（リスト6-15）
▼MEMOで作ったデータを，BASICを使ってグラフ化するプログラムも紹介しています（リスト5-13）

▶BOXEを使って描いた模様です．カラーパレットを使って，渋い色使いにしています（リスト6-12）
▲「スキー」の演奏とともに山が雪化粧していくカラオケソフト．ここでもカラーパレットが活躍しています（リスト5-4）
▼球が床に影をおとしている様子です．光線追跡法を利用して描いていますが，全部描くのに1時間24分かかりました（リスト6-22）

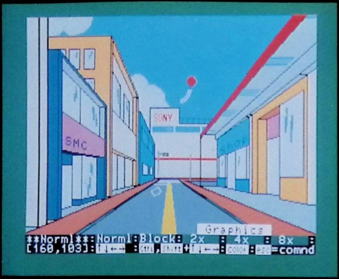






SMC Graphics Editor

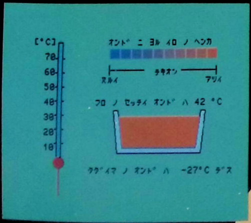
ソニーから発売されている「SMCグラフィックスエディター」を使えば，プログラムを作らなくても，好きな絵を描くことができます．BASICと組み合わせて使えば，ゲームを作るのも簡単



▲SMCグラフィックスエディターのエディットモードを使って，町並みを描いてみました

▲ファンクションキーを押すだけで，簡単に拡大してみることができます

▶BASICと組み合わせて作ったモグラたたきゲームの画面です（リスト6-21）



▲風呂の温度センサーを作ってみました．表示画面はSMC-777のグラフィックス機能をフルに活かしています

SMC-777は，マニュアルで全回路図が公開されています．I/O Expansionに自作のインターフェイスを接続して，応用周辺機器を作ることもできます

Hardware

SMC-777ガイドブック

更科潤二 & ANIMAL HOUSE 著

アスキー出版局

はじめに

本書はソニーのパーソナル・コンピュータ，SMC-777を材料として，あれこれと料理を試みた結果をまとめあげたものです．どんな料理もそうなのですが，材料のもち味に従って，いわゆる〝素材の良さを活かす〟ような調理法がベスト．本書もそのような手順を踏まえて書かれました．

一般的なパソコンの解説書が，比較的均等に各部の特徴を取りあげているのに対し，本書はSMC-777の際立った部分に焦点を絞って構成されています．それが材料のもち味を活かしたということで，特にDr. Logoとグラフィックスに力点が置かれています．前者のLogoは，これから注目を集めてゆくであろうLISP系の言語に対する入門的な意味あいで，また後者のグラフィックスは，映像に強いソニーならではの数々の機能を評価して，それぞれ表面的な説明のみに留まらず，少しばかり突込んだ解説を試みました．

すでにSMC-777に触れてみた方ならお気付きでしょうが，このマシン，非常にユニークな特徴を備えています．それはフロッピーディスクドライブ(FDD)内蔵であるとか，安価なカラーパレットの供給というようなハード的な側面ももちろんですが，むしろソフトも含めた設計思想そのものにあるようです．ビギナーからエキスパートまで，広範な層のユーザーがそれなりに使うことのできるパソコン── SMC-777をいじっていると，そんなソニーの技術者の設計思想が伝わってくるような気がします．確かにこのマシンは，どんなレベルのユーザーが使ってみてもレベルに応じた満足を与えてくれるようで，たとえば何の予備知識のない人でもとにかく使うことができるし，BASICなんぞもう飽きたというマニアでも，5番地・8番地のDOSコールを相手に結構あれこれ楽しめます．

本書はそんなSMC-777の〝良さ〟を前面に出し，ひととおりの全体像が把握できるように構成しました．ただ，SMC-777の広い守備範囲のすべてを，一冊の本という制限のなかで，何の予備知識もない方に紹介しつくすのは困難．そこでLogoに関する章は別として，他の章はそれぞれで解説するテーマに対するベーシックな知識を持った人を対象に記述されています．ですからBASICやグラフィックスについての章を読む前提としては，多少のBASICのプログラミングの知識が，ハードについての章を読む前提としては，若干のハードに関する素養が必要ということで，必要に応じて他の文献を参照してくださるよう，お願いしておきます．

ともあれ，将来のホーム・コンピュータの原型とも言えるこのSMC-777を，本書をきっかけとして多くの方に楽しんでいただければ幸いです．

1984年4月　著者代表　更科 潤二

本書を読む前に

本書は，ソニーから発売されたユニークなパソコンSMC-777をハード・ソフトに渡って総合的に解説したガイドブックです．SMC-777は従来のBASICマシンと違い，ソニー独自のオペレーティング・システム(Sony Filer)の下，各種のソフトウェアが標準搭載(バンドリング)されています．ここに本書の構成を示しますので，読者の方々の興味や知識に応じて，それぞれ好きな章から読み始めていただきたいと思います．

1章　SMC-777概説

パソコン後発メーカーとして，ソニーはSMC-777に様々なアイデアを盛り込んでいます．本章ではそうした特徴的な部分を，ハード・ソフト共に説明しています．また，FDDを1基内蔵したマシンとしてDOSを持つSMC-777では，特にOSたるSony Filerがユニークな側面をいくつも持っています．そこで，Sony Filerのオペレーションについて，本章では特にページを割いて解説してゆきます．

2章　Logo初級講座

MITのセイモア・パパートが生み出したLogoのソニー版，Dr. Logo*について概説してゆきます．Logoに込められた開発理念に始まって，Logoとイコールで結ばれている観もあるタートル・グラフィックスについて説明する本章は，Logoのイントロダクションに相当します．

なお，本書ではLogoのバージョン1.4を基本としています．バージョン1.0の読者の方は本書のとおり動かないこともありますので，バージョン1.4との交換について，マシンの購入店で相談してください．

3章　Logoプログラミング

タートル・グラフィックスの方法論が理解されたところで，Logoが単なるお絵描きツールではなく，立派なプログラミング言語であることを証明する意味で，いくつかのアプリケーションの作り方を紹介してゆきます．特にこれらのプログラミングを通じ，構造化プログラミングやプロシージャ，再帰呼出し（リカーシブコール）というような，Logo，ひいてはLISPに通じる概念を提示しています．

* Dr. Logoは米国デジタルリサーチ社の商標です

4章　Logoリスト処理

Logoの最大の特徴は，数値のみでなく記号も扱えるという，リスト処理の能力にあります．とりわけDr. Logoにはプロパティ管理の能力があるために，いわゆる人工知能的なプログラムの片鱗をうかがうことも可能です．本章ではこうしたリスト処理やプロパティ管理の初歩から，それらの能力を応用したプログラムの例示までを内容とし，ひと味違った言語としてのLogoを説明してゆきます．

5章　777 BASICへの招待

777 BASICは，一般的なBASICと比較していろいろと機能的に追加された部分があります．本章ではこうした777 BASICの諸特徴から始まって，比較的ユニークな特徴を備えたSOUND命令（マルチタスク的な使い方が可能）の実践的な使い方，ディスクまわりのコマンド等を解説．さらにバンドリングされている作表言語MEMOのBASICによる活用方法を解説してゆきます．

6章　グラフィックスの世界

SMC-777のグラフィックス機能には，他の機種とはひと味違った強力なものがあります．本章ではそんなグラフィックスに関して，基本となる知識から3次元グラフィックスまで，かなり詳細に説明してゆきます．本章は全体に渡り，オプションとして用意されているカラーパレット(4096色から16色を選択表示可能）の利用を前提としており，これについての活用法の説明も用意しました．このほかソニーより発売されているSMCグラフィックスエディターの紹介とそのBASICによる活用法も説明します．なお，グラフィックス関連コマンドを中心としたN_{88}-BASICからの移植テクニックの解説も，多くのユーザーに役立てていただけると思います．

7章　ハードウェア

SMC-777は，マニュアルで全回路図が公開されたマシンであり，アセンブラ等を使ってBASICやLogoでは実現できないマニアックな利用方法が実現可能です．本章では，このハードウェアの概要から始まり，メモリ構成，画面構成，割込みへと解説を進めます．なお，応用周辺機器の自作例として，風呂の温度センサーを紹介します．

8章　アセンブラとSony Filer

本章はバンドリングされた777 Assemblerを使って，Sony FilerやSMC-777のハードウェアの能力を十分にひき出したいと考えている方のために設けました．777 Assemblerの解説に始まり，DOSコールの説明とその具体的なやり方，さらにSony Filerへのコマンドの追加方法や割込みを利用したプログラム例などを示してゆきます．

APPENDIX

各章の本文では触れきれなかった，やや専門的な解説や，SMC-777ユーザーに有益な表，ギフト・プログラムなどを載せたのが，このAPPENDIXです．

APPENDIX 1では，Logoの本文では触れきれなかった解説や，Dr. Logoのすべてのプリミティブを載せた一覧表を掲載．そのほかにN_{88}-BASICで書かれたプログラムを777 BASICへ移植する際に役立つクイック・コマンドリファレンス，UNIXのテキスト表示コマンドであるmoreを真似たギフト・プログラム，ザイログ・ニーモニックからANN表記への対応表など，盛り沢山な内容となっています．

目　次

はじめに……3
本書を読む前に……4

1章 SMC-777概説……13

1-1 ———ハードとソフトの主な特徴……13
（1）ハードウェアの特徴……13
（2）ソフトウェア紹介……16
1-2 ———Sony Filerの概要……17
（1）Sony Filerの起動……17
（2）各種コマンドの役割りと使い方……22

2章 Logo初級講座……35

2-1 ———Logoの全体像……35
（1）卓抜した教育用言語としてのLogo……35
（2）進化した言語としてのLogo……37
（3）人工知能型言語としてのLogo……37
（4）Logoの位置付け……38
2-2 ———Logoに触れる前に……39
（1）プロンプト状態……39
（2）プリミティブ……41
（3）表示画面……43

2-3 ——タートル・グラフィックス入門 …… 47
(1) カメ，動け！ …… 47
(2) カメの縄張り …… 55
(3) カメ，働け！ …… 58

3章 Logoプログラミング …… 71

3-1 ——立体柱を描く …… 71
(1) まず四角形から …… 71
(2) プロシージャはエディットモードで …… 72
(3) 底面そして高さ …… 75
(4) ディスクへの保存と読込み …… 77
(5) Logoのプログラム構造 …… 79
3-2 ——変数と引数 …… 80
(1) 引数・変数・定数 …… 80
(2) グローバル変数とローカル変数 …… 81
(3) 値を返すプリミティブ …… 82
3-3 ——サンプル・アプリケーションズ …… 83
(1) 三角関数のワザ …… 83
(2) 数当てゲーム …… 89
(3) Logo Clock …… 90
(4) ドラゴン曲線を描く …… 95

4章 Logoリスト処理 …… 99

4-1 ——リスト処理関連のプリミティブ …… 99
(1) 要素関係のプリミティブ …… 100
(2) 加工関係のプリミティブ …… 103
(3) ワード処理関係のプリミティブ …… 105
(4) 調査役プリミティブ …… 106
(5) データ管理プリミティブ …… 108

4-2 ―――リスト処理サンプル・プログラム …… 111
(1) 電話帳プログラム …… 111
(2) 英文和訳プログラム …… 114
(3) 暗号とLOGOBASE …… 117
(4) まとめ …… 124

5章 777 BASICへの招待 …… 125

5-1 ―――777 BASICの特徴 …… 125
(1) 操作性 …… 125
(2) EDITとDEBUG …… 129
(3) 特殊な関数 …… 133
(4) リンクモジュール …… 134
(5) ファイルのロードとセーブ …… 135
(6) その他の主な特徴 …… 138
5-2 ―――効果音とサウンド …… 139
(1) コマンドと効果音 …… 140
(2) PLAYプログラミング …… 143
(3) SOUNDプログラミング …… 145
(4) 777オルガン …… 148
5-3 ―――ディスク関連のコマンド …… 151
(1) OPEN／CLOSE …… 151
(2) PRINT＃／INPUT＃ …… 152
(3) SEND／RECEIVE …… 153
(4) データファイルを扱うサンプル・プログラム …… 153
(5) MERGEとCHAIN …… 155
5-4 ―――BASICによるMEMOの活用 …… 157
(1) 作表 …… 158
(2) MEMOが作るファイル …… 159
(3) プログラム例 …… 163

6章 グラフィックスの世界……169

6-1 ――コンピュータ・グラフィックスの役割り……169
6-2 ――グラフィックスの基礎……170
(1) 画面構成……171
(2) キャラクタ表示……171
(3) グラフィック表示……173
(4) グラフィック座標系……173
(5) 三角関数……175
6-3 ――グラフィックス関連コマンド……176
(1) GPLOT……178
(2) LINE……179
(3) BOXE, BOXF……181
(4) CIRCLE……183
(5) DRAW……184
(6) その他のコマンド……189
6-4 ――カラーパレット・テクニック……191
(1) カラーパレットの基本……191
(2) 立体図形とグラデーション……192
(3) 動画……197
(4) 24色表示……198
6-5 ――グラフィックス上級編……201
(1) 線画による3-Dグラフィックス……201
(2) 光線追跡法(ray-tracing)……214
6-6 ――SMCグラフィックスエディターの利用……219
(1) 使い方……220
(2) BASICによりGEのファイルを読み込む……222
6-7 ――N_{88}-BASICからの移植テクニック……225
(1) 移植テクニックの基本……225
(2) サンプル・プログラム……227

7章 ハードウェア 235

7-1 ――ハードウェア構成 235
(1) 概要 235
(2) シャドウROM 236
(3) 画面 237
(4) 割込み 240
(5) SMC-70との違い 240
(6) その他 240

7-2 ――応用周辺機器の自作 241
(1) ハード 241
(2) 温度センサー用プログラム 245

8章 アセンブラとSony Filer 249

8-1 ――アセンブラ 249
(1) 777 Assemblerとは 249
(2) 実行例 255

8-2 ――DOSコール 261
(1) なぜDOSコールなのか 261
(2) サンプル・プログラム 262
(3) 5番地と8番地の違い 266

8-3 ――Sony Filerのコマンドを作る 266
(1) Sony Filerとは 266
(2) コーディングにあたって 267
(3) LOGOPRNT 269

8-4 ――割込み 273
(1) SMC-777における割込み 273
(2) 割込みを利用したサンプル・プログラム 275

APPENDIX

1 ——Logo …… 279
(1) エスケープ文字について …… 279
(2) 変数名 …… 280
(3) 演算子 …… 282
(4) catchとthrow …… 283
(5) システムプロパティとシステム変数 …… 284
(6) プリミティブ一覧表 …… 286
2 ——クイック・コマンドリファレンス …… 304
3 ——Assembler …… 307
(1) MORE …… 307
(2) ザイログニーモニックからANN表記への対応表 …… 311
4 ——各種コード表 …… 313

テクニカル&ライティング・スタッフ
- ANIMAL HOUSE(代表：更科潤二)
- GOD ROOM
- 埼玉工業大学　深町研究室

1章　SMC-777概説

ソニーSMC-777は，数々のユニークな特徴を持つパーソナル・コンピュータとして，発売以来，各方面の注目を集めているマシンです．このSMC-777は，ソニーが最初に発売した非常にマニアックなパソコンであるSMC-70の特徴を受け継ぎ（確かに機械としてはとてもよいパソコンだった……），さらにユーザーオリエンテッドな各種の工夫をこらして完成されたもので，良きにつけ悪しきにつけ，SMC-70の経験と反省の上に立脚した機種であると言えるでしょう．ソニー自身も販売に力をいれ，またユーザーの方でも期待を込めて見守るSMC-777には，148,000円という価格の中に可能な限り高水準な機能が盛り込まれている上に，マニアックなユーザーにもまるきりのビギナーにも，等しく満足を与えられるような配慮がいくつも込められています．本章では，このSMC-777の主だった特徴と，基本的な操作方法を紹介してゆきましょう．

1-1　ハードとソフトの主な特徴

(1) ハードウェアの特徴

3.5インチディスク内蔵

SMC-777の最大の特徴は，何と言っても本体内にフロッピーディスクドライブ(FDD)が1台，内蔵されていることでしょう．3.5インチのマイクロフロッピーディスクを外部記憶として使えるということで，このマシンの使い勝手は非常に高いものとなり，これはソニーがSMC-777を，ソフトウェアプレーヤーと呼ぶ理由となっています．少なくとも，カセットテープレコーダを用いてのロードやセーブの面倒臭さ（あの待ち時間は苦痛以外のなにものでもない）に泣いたり，オプションのFDDの痛い出費に泣いたりからは解放されるのです．しかも，片面ながら280Kバイトと，両面倍密のミニフロッピーディスクとほぼ同じ容量を持つこのディスクは，アクセス速度となると5インチのものよりはるかに速く，高速記録／読み出しを実現しています．

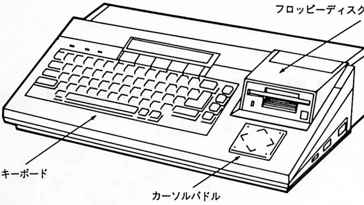


図1-1　SMC-777 外観図

またソニーの3.5インチディスクはプラスチックのジャケットに覆われ，ヘッド·ウィンドウの保護を目的としたシャッターも付いています．このシャッターはディスクをFDDに出し入れするときに自動的に開閉するので，初心者でもディスクを事故なく扱えるというメリットを生んでいます(余談ですが，3.5インチのフロッピーディスクを郵便でそのまま送っても，何も問題がありませんでした．5インチ，8インチではそうはいくまい……)．本体FDD内蔵がSMC-777の前提なのですから，OSであるオリジナルのSony Filerも，ディスクアクセスを十分に考えた上で作られていることは，言うまでもありません（システムメニューの表示が，そのよい例です）．

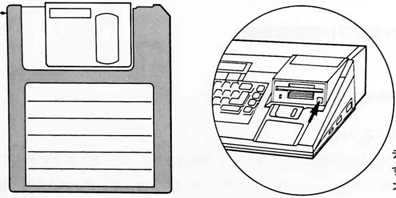


図1-2　3.5インチフロッピーディスク

カーソルパドル

FDDとともに外観の大きな特徴となっているものに，従来のカーソルキーに代って装備されたカーソルパドルがあります．センターに支持点を持ち，二重接点のキーが合計4箇所に配置されているこのカーソルパドルは，ジョイスティック的な感覚をカーソルキーに持ち込んだものであり，メニュー画面にあってはマウスのようにポインティングをするときのツールとしての役まで務めます（ただし，機械精度はそう高いものではないため，故障の危険は少々あります）．

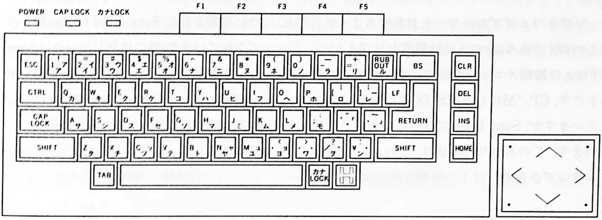


図1-3 キーボードの配列

キーボード

キーボードは，キー配列がセレクトリック・タイプライタのもので，JISの配列とは少々異なっており，[：]，["]，[＝]といった記号の位置が，大型機の端末と同じようなスタイルで配列されています．また，カナ文字の配列もJIS方式とは大幅に異なり，アイウエオ順となっているため，カナ入力に慣れない人には便利です．

キーの並び具合も他の機種とは異なり，隣りのキーと少々離れているために，初めのうちはとまどいを感じることもあるでしょう．しかし，慣れてしまうとこちらの方が好ましく思ったりするもので，また押し勝手も適度な手応えを持つため，ミスタッチの回避などに役立っているようです．

カラーパレット

カラーパレットボードを取り付けることにより，SMC-777のカラー表示能力を4096色まで広げることができます．カラーパレットボードを取り付けると，これまで標準16色しか表示できなかったものが，4096色より16の色を自由に選べるようになり，微妙な色あいを表現できるようになるわけです．

このカラーパレットボード（SMI-733）はオプション（別売り）ですがそれほど高価なものではなく，SMC-777においては準標準装備と考えられています．ただし，このカラーパレットの能力を生かすにはアナログRGB端子の付いたモニタが必要であり，この種のモニタは今のところあまり種類がない（もちろんソニーからは発売されています）ので注意が必要です．

(2) ソフトウェア紹介

ソフトウェアプレーヤーと自称するように，SMC-777に搭載されたSony Filerは使い勝手のよいOSであり，その上には豊富なソフトがバンドリングされています．特に注目したいのはSony Filerの初期メニュー画面で，他のOSには見られない，簡易言語のメニューを思わせる使い易さです．CP／MにしろMS-DOSにしろ，しょっぱなから難解なイメージを与える画面構成になっていますが，Sony Filerではとっつき易さを感じさせ，誰でも使えるOSといったものに仕上っています．このあたりの配慮は，これからのOSのひとつの行き方にさえなるかもしれません．バンドリングされたソフトの詳細な解説は各章に譲るとして，ここでは概略を説明しておきましょう．

777 BASIC

言わずと知れたBASICなのですが，パソコン後発のソニーらしく，様々な機能を加えて搭載しています．全体的に操作性がよく，特にデバッグのし易さへの配慮は注目に値します．この他の主な特徴としてテキスト画面では，各文字ごとにアトリビュート指定（各文字ごとに表示色と，バックグラウンドカラー，ブリンクの有無を指定できる）ができること，キャラクタジェネレータがすべてプログラマブルであることなどが挙げられます．サウンド機能として，SOUNDコマンドが，実行の終了を待たずに次に続くコマンドを実行するマルチタスク的な機能を持つことから，音と画面の併行出力が可能なことも見逃せない点であり，またユニークなDRAWコマンドを始めとした豊富なグラフィックス機能により，高度なグラフィック処理も可能となっています．

Dr. Logo

スイスの児童心理学者，ピアジェの発達心理学を下敷に，MITのパパート等が創り出した人工知能言語，Logoは，いま各方面で注目されているコンピュータ言語です．ひとつには新しい教育の概念を持ち込み，子供の創造性を高めるための教育がLogoを通じて行えることから．また，もうひとつは，人工知能言語として名高いLISP (List Processor) の機能をそのまま受け継いだ言語として，リスト処理や再帰呼出し，ワード，オブジェクトといった概念をLogoによって学べることから，次世代のプログラミングへの導入としてLogoが位置付けられているのです．

Assembler & Debugger

機械語プログラミングに欠かせない，アセンブラとデバッガがバンドリングされています．アセンブラの方はBASICのエディタを用いてソースリスト（ソニー独自のANN表記）を作るため，初めての人にもとまどいなくソースプログラムを作れるという利点があり，またデバッガの方もかなり拡張されたコマンドを持ち，強力なデバッギングツールとなっています．

MEMO

いわゆるスプレッドシート型の簡易言語MEMOは，実用性の高いアプリケーションです．簡単なメモ帳やアドレス帳的なものから，集計用紙や帳簿として使えるまでの機能を持ち，容量的にもかなりのものであるために広い使途が考えられます．少なくとも市販のこの類の簡易言語のなかでもMEMOに遅れをとるようなシロモノがずい分とあることから，バンドリングソフトとしては上々のできであると言えるでしょう．

Games

3つのゲームが搭載されており，プログラミングの息抜きや時間つぶしには，それなりに思わぬ楽しみを味わうことができます．

以上がSMC-777のハードとソフトの概略ですが，本書の以降の章で，各々の詳細を追って解説してゆきます．ある意味では日本のAppleIIになり得るパフォーマンスを持ったマシンですので，皆さんが積極的にチャレンジされることを期待します．

1-2　Sony Filerの概要

SMC-777がどのような性格を持つマシンであるかは，いく分おわかりいただけけたと思います．続いて，ハードウェアと共にSMC-777の機能の柱となるOS（オペレーティング・システム），Sony Filerについて，さらに解説を加えます．

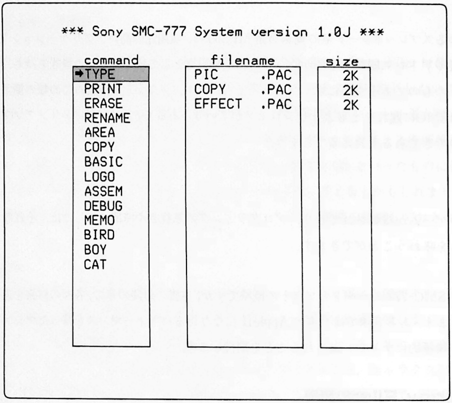
Sony Filerはシステムメニューと呼ばれる初期画面を持ち，登録されているコマンドやファイルはこのメニューを通じて選択されるように工夫されています．本節ではSony Filerの起動方法や各種コマンドの使い方について触れることで，このSony Filerの使いやすさと機能を見ていくことにしましょう．

なおSony Filerの構造やファンクションコールについては，8章を参照してください．

(1) Sony Filerの起動

さっそくSMC-777を動かしてみることにしましょう．まずモニタの電源，続いて本体裏面の電源スイッチをONにします．ピッと音がして本体上面のパワーインジケータランプが点灯しますので，次にシステムディスクを本体右側のFDD（フロッピーディスクドライブ）に入れると，ランプが灯いてディスクが回転を始めます．

モニタにSony Filerのシステムメニューの画面が出ましたか？　ではコマンドの起動方法から説明してゆきましょう．



画面1-1　システムニュー画面

コマンドの起動方法

従来のDOS（ディスクオペレーティング・システム）と異なり，Sony Filer ではシステムメニューの画面上でのコマンドやファイルの起動は，カーソルパドル（カーソルを動かす矢印の付いている板状のスイッチ）で行えます．試しに，[↓]，[↑]の方向にカーソルパドルを押すと，[⇨]が TYPE, PRINT ……の項に移ってゆきます．次に[→]の方向に押すと filename の欄に[⇨]が移り，[←]の方向に押せば，[⇨]は command の欄に戻ります．この機能を使って，コマンドやファイルを選んで使うわけです．

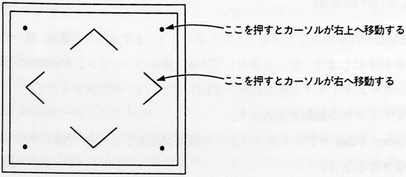
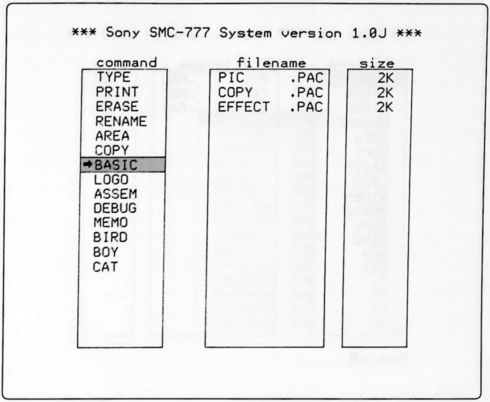
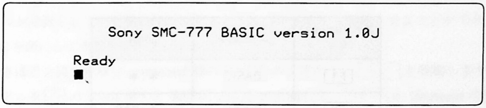


図1-4　カーソルパドル



画面1-2 カーソルパドルでBASICを選んだところ

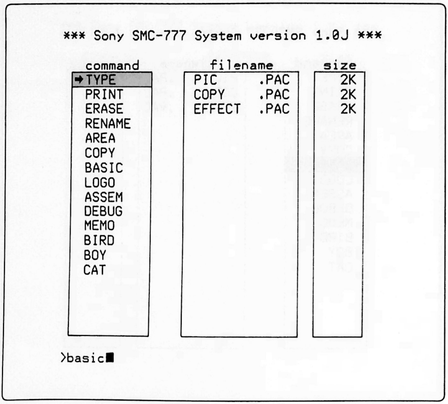


画面1-3 BASICの初期画面

試しにお馴染みのBASICを起動してみましょう．command欄にある[⇨]をカーソルパドルを使ってBASICのところに移動させ，RETURNキー（以後[↵]）をポンと押すだけでOK．システムメニューのBASICの文字が点滅して，画面の下のほうに…loadingというメッセージが表示され，FDDが動いてBASICを読み込みます．画面が黒に変り，「Sony SMC-777 BASIC version ×××」「Ready」のメッセージが出て，■マーク（カーソル）が点滅すればBASICが入力待ちの状能であることを示しています．

システムメニューからのBASICの起動は，[⇨]を動かす以外にも方法があります．キーボードからBASICと入力してください．このとき小文字で入力しても大文字で入力しても問題ありません．画面の下に> BASICと表示されます．キーボード上部に付いているファンクションキーの[F1]を押しても同じ結果です．習うより慣れろ．試しにキーをたたいてみてください．

BASICに限らず他のコマンドもこの方法で起動することができます．



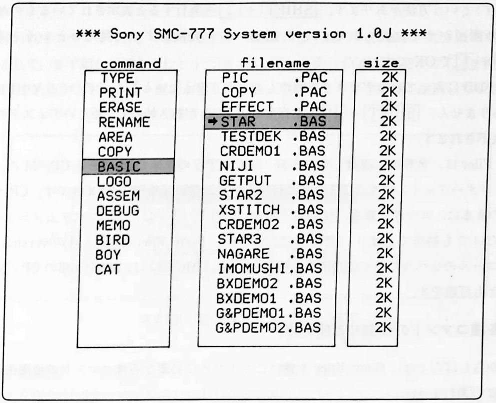
画面1-4　キーボードから直接BASICと入力する

ファンクションキー	ノーマル	Shift
F1	BASIC	*.*
F2	LOGO	*.BAS
F3	PRINT	*.LOG
F4	TYPE	*.TXT
F5	COPY	*.ANN

表1-1　システムメニュー画面時のファンクションキーの働き

filename欄の使い方

次に，右隣りのfilename欄の使い方を説明しましょう．初期設定されたままのディスクにはまだファイルがセーブされていませんが，自分で作ったファイルや他のディスクからコピーしたプログラムは，この欄にファイルネームが，さらに右隣りのsizeの欄に各ファイルの容量が表示されます．ファイルの領域はディスク上2Kバイト単位で取られますので，この欄の表示も2Kバイト単位になっています．



画面1-5　BASICコマンドと共にSTARというファイルを選ぶ

ファイルが登録されている場合は，コマンドと共にファイルも起動させることができます．再びBASICを例にして実行してみましょう．

まずさきほどと同じ手順でcommand欄のBASICの文字の上に[⇨]を移動します．次にカーソルパドルを[→]の方向に押すとBASICの文字が点滅して，[⇨]がfilenameの欄の一番上に移動します．あとは起動させたい（プログラムの）ファイル名の上に[⇨]を持ってゆき，[↵]を押せばBASICが起動して指定したプログラムが自動的にスタートします．ただし，BASICのコマンドと同時にBASICとは無関係のファイルを指定すると，とんでもないことが起こる可能性があるので十分な注意が必要です．

なお，[⇨]がfilenameの欄にあるときに，[HOME]キーを押してからシフトキーとファンクションキーを同時に押すことで，LogoやBASICのファイルだけを表示できます．たとえば，[SHIFT]+[F2]を実行すると，filename欄には拡張子がBASのファイルだけが表示されます．また，表示を戻すときには，キーボード右下にあるヘルプキー（[H]）を押せばOKです．

さてここで先々のことを考えて，知っておくと便利な機能を説明しておきましょう．まずファイルを登録し過ぎてシステムメニューに収まりきらなくなったとき．このときは[⇨]をどんどん下げてゆくと，見えなかったファイル名が下から湧き出してきて上のファイル名が消えてゆきます．この機能があるのでファイルの数がいくら増えても心配はいりません．しかし，これではまだるっこいという人のためには，[SHIFT]＋[↓]（シフトキーを押しながらカーソルパドルの下

方向を押す）という方法があります．SHIFT ＋ ↓ を実行すると表示されているものがすべて消え，次の画面がすぐに見られるのです．同様に前の画面がすぐに見たいというときには，SHIFT ＋ ↑ でOKです．

次に，FDDに入っているディスクを別のものに取り換えたいときは，そのつどリセットをかける必要はありません．SHIFT ＋ H を実行するとFDDが動き始めて，新しいディスクの内容が画面に表示されます．

Sony Filerは，世界的に流通している8ビットCPUの標準DOSであるCP／Mの，マンマシン・インターフェイスの改良版といった性格を持つ，使い勝手のよいDOSです．CP／Mの機能をそのままに，ユーザー尊重，使い易さ重視の設計ポリシーは，このシステムメニューの画面を見ただけでも納得できます．余談になりますが，Sony FilerはCP／Mのversion 1．4とBDOSコールのレベルで多くの互換性を持ち，さらにSMC-777は，SMC-70用のCP／Mを使用することも可能です．

(2) 各種コマンドの役割りと使い方

ここからしばらくは，Sony Filerを使いこなすために必要な各種コマンドの使用法を解説することにしましょう．

単独で使用するもの	ファイルを必要とするもの	どちらでも使えるもの
AREA	TYPE	BASIC
COPY	PRINT	LOGO
MEMO	ERASE	DEBUG
BIRD	RENAME	
BOY	ASSEM	
CAT		

表1-2　Sony Filerのコマンド一覧

Sony Filerには，14のコマンドが入っています．このうちBIRD, BOY, CATの3つはSonyオリジナルのゲームです．『GAMES』というマニュアルをよく読んで遊んでみてください．なかなかよくできています（と思う……）．この他に，言語関係のコマンドとして，御存知のBASIC，注目の新言語LOGO，機械語関係のASSEM, DEBUGがあります．ファイル管理用のコマンドとしてはERASE, RENAME, TYPE, PRINT，ディスク管理用にはAREA, COPYが用意されており，さらに，簡易言語の機能を持つMEMOもあります．言語関係のコマンドの使い方は，本書で詳細に解説されていますので，それぞれの章を読んでください．ここでは，初心者の方を対象にして，ファイル関係とディスク関係のコマンドについて，実習しながら解説してゆくことにしましょう．

COPY

よく使われるコマンドです．本書のモットーは「習うより慣れろ」．細かいことは順々に説明することとして，まず動かしてみましょう．システムメニューからCOPYを選んで ↵ を押してください．COPYコマンドメニュー画面が出てきます．では，よく使う順番で説明してゆきます．

```
*** Sony SMC-777 Copy version 1.0J ***

[ F1 ]  ➡Copy disk files
[ F2 ]   Copy disks
[ F3 ]   Format disks
[ F4 ]   Convert cassette files
[ F5 ]   Generate system disk
[HOME]   End of copy
```

画面1-6　COPYコマンドのメニュー画面

● Format disks

これは，左側の［F3］という表示からもわかるとおり，ファンクションキーの F3 を押しても使うことができます．買ってきたばかりの新品のディスクは，磁気面もまっさらで何も入っていません．これを，コンピュータが使える初期状態にするため，ディスク内を区分けしたりファイルを書き込む番地を付けてやったり，いわば真っ白な紙の上に文書を書き易いように罫線を引いてやることを，フォーマッティングと言います．このフォーマッティングを行うコマンドがFormat disksです．これは，さきほどのまっさらな新品ディスクのほか，一度使っていろいろ書き込まれているが中身は必要でないディスクを初期化して使うときにも用いられます．では，講釈はこれくらいにして，実際に使ってみましょう．

まず新品のディスクか，不用になったディスクを1枚用意してください．COPYメニュー画面からカーソルパドルを使ってFormat disksを選ぶか，あるいはファンクションキーの F3 を押してください．すると，Format disksの文字が点滅し，画面下に，

Are you sure ?　(Y/N)

という表示が出ます．フォーマットすると，ディスクの内容がすべて破壊されてしまうので，「間違いないかね？」とたずねてくるわけです．ここで Y をキーボードから入力すると，

Insert NEW disk in SYSTEM drive :

という表示が出ますから，フォーマットしたいディスクをFDDに入れ， ↵ を押します．するとフォーマッティングが始まり，現在フォーマットされているトラック№が1，2，3……69と順に表示され，次に69,68,67……1とVerifyしていきます．Verifyというのは，確認という意味で，

各トラックがきちんとフォーマットされたかどうか確かめているわけです。

もし，間違ってこのコマンドを選んでしまったり，フォーマットしたいのにCopy disksを選んでしまったら．こんなときは，キーボード左上の[ESC]キーを押せばOK．COPYのメニュー画面に戻ってくれます．

● Generate system disk

フォーマットが終ったら，次にこのディスクにシステムを書き込みます．システムとは，Sony Filerを起動させたり，Sony Filerを動かすためのプログラムです．

ではファンクションキーの[F5]を選んでください．画面下に，

```
Examine NEW disk ? (Y/N)
```

と表示されます．

まずは，[N]から選んでみましょう．

```
*** Sony SMC-777 Copy version 1.0J ***

[ F1 ]   Copy disk files
[ F2 ]   Copy disks
[ F3 ]   Format disks
[ F4 ]   Convert cassette files
[ F5 ]   Generate system disk
[HOME]   End of copy


    [1]   Single drive
    [2]   Dual   drive
    [3]   Single drive (Auto-Start)
    [4]   Dual   drive (Auto-Start)

          Select [1..4] :
```

画面1-7 Generate system diskで[N]を選んだときに出る画面

また，メニューが出てきました．Sony Filerでは，FDDを本体内蔵FDD 1台で使うのか，増設FDDをつないで2台で使うのか，あるいはAuto-Start（後述）で使うのかを選択することができます．これを数字キーの1から4で選択するわけです．

[1]は，Single drive．本体内のFDDだけを使うシステム用です．[2]は，Dual drive．本体内FDD＋増設FDD用．[3]は，Single driveで使うAuto-Start用．[4]は，Dual driveのAuto-Start用です．

Auto-Startについては，少々説明が必要でしょう．通常，Sony Filerではメニュー画面から

各コマンドを選択して起動させます．しかし，Auto-Start設定をしておけば，電源を入れてディスクを差し込むだけで，メニュー画面を飛ばして目的のコマンドを使うことができます．たとえば，BASICをAuto-Start設定したディスクを作っておけば，ディスクを差し込むだけですぐにBASICが使えるわけです．

では，BASICをAuto-Start設定にしたディスクの作り方を実習してみましょう．まず，

Single-drive（Auto-Start）

を数字キーの[3]で選択します．すると，下に，

Enter Auto-Start Command Line

と表示されますので，ここではBASICと入力して[⏎]を押します．画面の表示に従ってシステムディスク（Sony Filerの入っているオリジナルディスク）を入れて[⏎]．次にさきほどフォーマットした新品のディスクを入れて[⏎]．あとは，このディスクに入れるコマンドを一つひとつ聞いてくるので，[Y]／[N]で選択すればよいわけです．[BASIC. EXE]は，もちろん[Y]．その他に[AREA.EXE]，[COPY.EXE]の2つは，入れておくと便利です．これで，BASICのAuto-Startディスクができあがり．もし，ディスクの枚数に余裕があれば，LOGOとMEMOの専用Auto-Startディスクを作っておいて使い分けると，賢いSMC-777ユーザーと呼ばれることになるでしょう（？）．他の，Single drive用，Dual drive用でも使用方法は同じです．最初に出てきた「Examine NEW disk （Y／N）」で[Y]を選択しても，画面のメッセージに従えば使い方の基本は同じです．

Dual driveを選択して2ドライブ用のディスクを作った人は，ちょっと通常とは使い方に異なる点があるので注意が必要です．

まず，本体内のFDDだけではDual drive用のディスクは起動しません．たとえ増設FDDは使

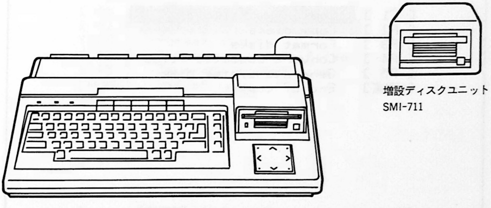


図1-5　増設ディスクユニットの接続図

用しなくても，何かディスクを入れておいてやる必要があります．しかし，これにも抜け道があり，増設FDDがつながっていなかったり，ディスクが入っていなかったりすると，画面下に，

Please Insert Disk and Hit [RETURN]：

という文字が点滅します．このとき，ESC を押すと，

User drive offline. O.K. (Y／N)？

に表示が変ります．「増設FDDは，使わなくていいんだね」と聞いてくるわけです．ここで，Y と入力すれば，Dual drive用のディスクも電源を切るまでSingle drive用と同じに使うことができるわけです．

● Copy disk files

これを選択すると，

Filename：

と表示されます．これは，ファイルだけをコピーするコマンドなのです．

使い方はいたって簡単．コピーしたいファイル名を入力して，あとは表示に従ってディスクを入れ替えて ⏎ を押してやればOK．ここでも便利な使い方をお教えしましょう．Sony Filerは，さきほどお話ししたとおり，CP／Mと類似した性格を持つためCP／Mの便利な使い方がそのまま使えるのです．たとえば，BASICのファイルだけを全部コピーしたいというような場合，Filename：のところに「＊．BAS」と入力してやればOK．［＊］は，アスタリスクと呼ばれる

```
*** Sony SMC-777 Copy version 1.0J ***

[ F1 ]  Copy disk files
[ F2 ]  Copy disks
[ F3 ]  Format disks
[ F4 ]  Convert cassette files
[ F5 ]  Generate system disk
[HOME]  End of copy




Filename :     ■
```

画面1-8　Copy disk filesの画面

記号で，数値演算の乗算の記号ですが，ここではファイル名の代わりに使用し，すべてのという意味を持つファイルマッチ記号です．また，ファイルマッチ記号に[？]を使うこともできます．これは，[？]がその位置における1文字に相当します．たとえば，「A？？？．BAS」と入力すれば，Aで始まり，全部で4文字のBASICのファイルをすべて探し出してきてコピーしてくれます．たくさんのファイルをコピーするときなどとても便利です．

Copy disk filesでファイルをコピーするとき，指定したファイル名がすでにBackup disk上に存在していると，

File exists. Erase (Y／N)

つまり，「このファイルはもう書き込まれているけど，古いファイルは消してもいいのかね？」とたずねてきます．新たに書き込みたいときは，[Y]を押せば古いファイルは消去されて新しいファイルが書き込まれますし，[N]を押せば，そのままコピーのメニューに戻ります．

●Copy disks

次に，ディスクをまるごとコピーしたい，というようなときに使うメニューを紹介しましょう．

ファンクションキーで[F2]を選択してください．画面の表示に従って，コピーしたいもとのディスクを入れて[↵]を押すと，FDDが作動して，まず14トラック分読み取ります．次に，バックアップディスクを入れて[↵]すると，その14トラック分のデータが書き込まれ，さらにベリファイします．この動作を5回繰り返すと，そっくり同じ内容を持ったバックアップディスクが1枚でき上ります．

ディスクの入れ替えが面倒臭いな，と思われる方もいるでしょう．何度も入れ替えを行ってい

```
*** Sony SMC-777 Copy version 1.0J ***

[ F1 ]   Copy disk files
[ F2 ]   Copy disks
[ F3 ]   Format disks
[ F4 ]   Convert cassette files
[ F5 ]   Generate system disk
[HOME]   End of copy



Insert SOURCE disk :
```

画面1-9 Copy disksの画面

るうちにソースディスクとバックアップディスクを間違えて大切なファイルを壊してしまった，などということにもなりかねません．こんなとき，増設ドライブを使うとディスク入れ替えの手間がかからずとても便利です．

2ドライブで使うとき，画面のメッセージがちょっと違ってきますので注意してください．System driveというのが本体内蔵FDD，User driveが増設FDD側を示します．なお，老婆心ながら付け加えておくと，メッセージのなかで使われる，「SOURCE disk」，「SYSTEM disk」というのが，もとになるディスク，つまり読み取り用ディスクで，「BACKUP disk」，「NEW disk」が新たに書き込みをしようとするディスクを指します．念のため．

これで，ディスク同士のコピーの方法は理解できたでしょう．ディスクドライブの使い方に慣れていない方は，練習用のディスクを作って何度かトライアルをしてから本番にのぞまれるのがよろしいかと思います．大切なファイルを破壊してあとで泣かないためにも……．

●Convert cassette files

Sony filerでは，今までカセットテープを使っていた方のために，カセットテープからディスクにファイルをコピーするコマンドも用意されています．

ファンクションキーで[F4]を選択してください．ただしひとつ注意しなければならないことがあります．ここで扱えるファイルは，BASIC以外のファイルである，ということです．試しに，カセットテープからBASICのファイルを読み込んでみますと，

Skip ○○○○．BAS　(USE BASIC！)

と表示されて，BASICのファイルを素通りして次のファイルへ進んでしまいコピーはできませ

```
*** Sony SMC-777 Copy version 1.0J ***

[ F1 ]   Copy disk files
[ F2 ]   Copy disks
[ F3 ]   Format disks
[ F4 ]   Convert cassette files
[ F5 ]   Generate system disk
[HOME]   End of copy



Mount cassette and start :
```

画面1-10　Convert cassette filesの画面

ん．これは，BASICの中に「CLOAD」というカセットテープのファイル専用のコマンドがあるので，CLOADを使ってファイルを読み込み，それをSAVEでディスクにコピーしてくれ，という意味なのです．

では，BASIC以外のファイルの入ったカセットテープを用意して実行してみましょう．Convert cassette filesを実行すると，

Mount cassette and start :

という表示が出ます．カセットテープをセットしてから[↵]を押して，次にカセットテープレコーダをスタートさせます．カセットテープからファイルの読み込みが終了したらメッセージに従ってバックアップディスクに入れ替えて[↵]すればOK．Saveが終ると，

Convert more files ? (Y／N)

と表示されるので，まだ続けるなら[Y]を，もう終りなら[N]を押せばよいわけです．

これで，COPYコマンドの使い方はすべて説明しました．よく使うコマンドなので，しっかり慣れておいてください．最後に，終了の仕方．COPYのメニュー画面の一番下にある，

End of copy

をカーソルパドルで選ぶか，キーボード右側にある[HOME]キーを押せば，Sony Filerのシステムメニューの画面に戻ります．

ERASE

さて次は，いらなくなったファイルをディスク上から消してしまおう，というコマンドです．このコマンドは，Sony Filerのシステムメニュー画面上だけで用が足りてしまいます．では，例によって習うより慣れろ．

まず，コマンド欄の[⇒]をERASEの上に持ってゆきます．次に，[⇒]をfilename欄の消したいファイル名の上に移動させて，[↵]を押します．ERASEとファイル名が点滅して，下のほうに，

Erase ? (Y／N)

と表示されます．「消してもいいの？」と親切に聞いてくるのです．よく確かめて[Y]を押すと，FDDが作動して，画面上から指定したファイル名が消えてしまいます．

```
*** Sony SMC-777 System version 1.0J ***

 command       filename         size
 TYPE          PIC      .PAC     2K
 PRINT         COPY     .PAC     2K
 ERASE         EFFECT   .PAC     2K
 RENAME        STAR     .BAS     2K
 AREA          TESTDEK  .BAS     2K
 COPY          CRDEMO1  .BAS     2K
 BASIC         NIJI     .BAS     2K
 LOGO          GETPUT   .BAS     2K
 ASSEM         STAR2    .BAS     2K
 DEBUG         XSTITCH  .BAS     2K
 MEMO          CRDEMO2  .BAS     2K
 BIRD          STAR3    .BAS     2K
 BOY           NAGARE   .BAS     2K
 CAT           IMOMUSHI .BAS     2K
               BXDEMO2  .BAS     2K
               BXDEMO1  .BAS     2K
               G&PDEMO1 .BAS     2K
               G&PDEMO2 .BAS     2K

Erase ? (Y/N)
```

画面1-11　ERASEコマンドの実行画面

このほかにファイルを削除する方法としては，BASICのコマンドを使う方法があります．BASICの命令の中にERAというコマンドがありまして，マニュアル風に書くと，

ERA "[〈ドライブ名:〉] 〈ファイル名〉. 〈拡張子〉"

となります．本体内蔵FDDを使用する場合，ドライブ名は省略可能です．増設FDDのファイルを削除するときは，「B：」と入力してください．あとは，ファイル名と拡張子を入力してやれば，ERASEと同様にディスク上からファイルが消えてなくなります．

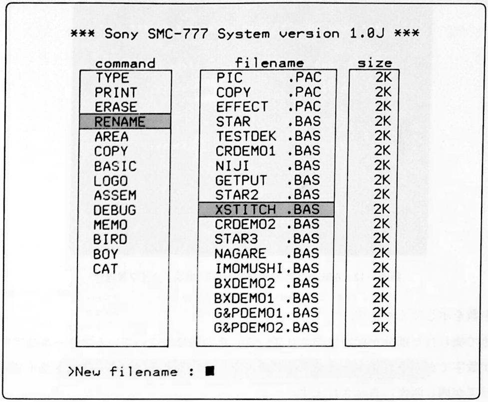
RENAME

これは，プログラムの内容を変更せずに，ファイル名だけを書き換えるコマンドです．

まず，command欄からRENAMEを選んで，次にfilename欄から変更したいファイル名を選択します．画面下に，

＞New filename：

と表示されますので，変更する新しいファイル名を入力してください．このとき，拡張子BASや，PACのようにファイル名のあとのほうに付いている拡張子（タイプ名）を忘れるとそのまま拡張子なしで登録されてしまいますから，忘れずに拡張子も入力してください．



画面1-12 RENAMEコマンドの実行画面

RENAMEでは，すでにディスク上に登録されている名前に変更しようとすると，

File exists

と，エラーメッセージを出してきますので，[ESC]キーを押して別の名前を入力してください．

さきほどのERASE同様，BASIC上でもファイル名の書き換えをすることができます．RENというコマンドがそれです．マニュアル風に書くと，

REN "[<ドライブ名：>] <旧ファイル名>.<旧拡張子>" TO
"[<ドライブ名：>] <新ファイル名>.<新拡張子>"

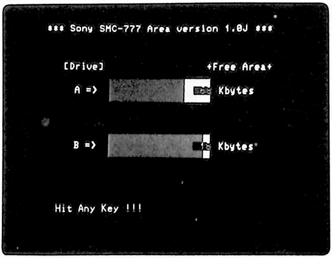
となります．試しに，JUN.BASをKEN.PACに変えてみましょう．

REN "JUN.BAS" TO "KEN.PAC"

これで[↵]すればOKです．

AREA

これは，ディスクの書き込み可能容量を知るためのコマンドです．Sony Filerのシステムメニュー画面からAREAを選んで[↵]を押すだけでOK．従来のDOSと異なり，ビジュアルに残り



画面1-13　AREAコマンドの実行画面(増設ドライブ接続時)

のバイト数を示してくれます.

赤い色で塗られた部分が使用中のエリア，青い色の部分が空いているフリーエリアで，残りバイト数が数字で表示されます．2ドライブで使用しているときは，A，B両ドライブの使用，未使用エリアが同じ画面に表示されます.

Hit Any key！！！

という画面表示に従って，任意のキーを押すとSony Filerのシステムメニュー画面に戻ります.

SMC-777で使われている3.5インチのマイクロフロッピーディスクは，全部で280Kバイトまでデータを書き込むことができます．しかし，ファイルの数が多くなってくると思わぬところで,

Out of disk space

などというエラーメッセージを頂戴することになるので，たまにAREAコマンドで残りのフリーエリアを確認する習慣をつけておくとよいでしょう.

TYPE

ファイルの内容をCRT上に表示するためのコマンドです．このTYPEコマンドで扱えるファイルは，アスキー形式で記憶されているファイルだけなので，通常のBASICのプログラムなどは表示できません．アスキー形式のファイルとしてよく見かけるものには，タイプ名に，ANNが使われているANN表記で書かれたアセンブラのプログラムや，SAVE／Aで記憶されているBASICのプログラムなどがあります.

使い方は，Sony Filerのシステムメニュー画面上で，TYPEコマンドと内容を見たいファイル名を選んで［↵］を押すだけ．途中で止めてよく見たいというときには，［CTRL］＋［S］を押し，実行を再開したいときには［ESC］以外の任意のキーを押せば始まります.

```
        make "eol :noe - :d + 1
        label "12
          make "j :i
          label "13
          make "t1 thing (word :j)
          make "t2 thing (word :j + :d)
          if :t1 > :t2
           [make (word :j) :t2
            make (word :j + :d) :t1
            make "j :j - :d
            if :j > 0 [go "13]
           ]
          make "i :i + 1
        if :i < :eol [go "12]
      if :d > 1 [go "11]
      ;
      make "i :noe
      make "l []
      repeat :noe
       [make "l fput thing (word :i) :l
        make "i :i - 1]
      op :l
      end

   Hit any key :
```

画面1-14　TYPEコマンドの実行画面

PRINT

このコマンドは，TYPEと同様にアスキー形式のファイル内容を表示する働きをしますが，CRT上に表示するのと同時にプリンタにも内容をプリントアウトします．

もし，プリンタがつながっていなかったり，プリンタに異常があると，

Printer not ready :

とメッセージが出るので，ESC を押して，一度Sony Filerのシステムメニューに戻ってからやり直してください．

SMC-777の接続方法に始まって，起動の方法からSony Filerのコマンドの使い方まで詳しく説明してきましたが，おわかりいただけたでしょうか．本章では，初心者の方々を対象に説明したので中，上級者の方々にはちょっと物足りない感を与えてしまったかもしれません．しかし，SMC-777は，従来の同価格帯のマシンの常識を打ち破った新機軸のFDD標準装備，DOS搭載，強力グラフィックス，etcが盛り込まれ，十分に楽しめるパソコンであることは確かです．また，マニュアルにハードウェアの詳細から回路図までも公開されるなど，ユーザー本位の姿勢がうかがえます．

初めてパソコンに触れる人のためには，楽しいゲームやLogoのタートル・グラフィックス，さ

らに使い易い777 BASICなども用意されています．本書の各章では，わかり易い説明と豊富なプログラムを用意し，どこから読み始めてもよいように構成を工夫しましたので，初めての方も臆せずキーボードをたたき，一日も早くSMC-777を使いこなしていただきたいと思います．

2章　Logo初級講座

SMC-777に標準搭載(バンドリング)された言語のうち，最も興味深いものがDr. Logoでしょう．周知のとおり，人工知能言語もしくは教育用言語としてつとに名高いLogoは，1967年にセイモア・パパート(Seymour Papert)教授の指導の下，MIT（マサチューセッツ工科大学）で産声をあげたコンピュータ言語です．Logoの開発にはMITの人工知能（AI = Artificial Intelligence）研究グループが全面的なバックアップを与えたと言われており，Logoの言語構造のいたるところに彼らの研究の足跡を見ることができます．そうした意味も含め，本章ではLogoとはいったいどんなものであるかを，その背景，予備知識，実際のオペレーションの順に説明してゆくことにしましょう．

2-1　Logoの全体像

（1）卓抜した教育用言語としてのLogo

Logoの存在を高名なものとした要因のひとつとして，教育用言語としての特性が挙げられます．コンピュータ，特にパソコンを用いた幼児教育は近年では非常に盛んになりつつあります．しかしそうした教育プログラムの大半は，近い将来の偏差値偏向教育の興隆を見越してか，子供をパブロフの犬に見たてたかのように，条件反射的学習に重きを置いたものばかり――子供の創造性であるとか自発的な探究心など，どこかに置き忘れられた観さえあります．このことは，ひとつには，子供に対してコンピュータが絶対的な権威者となり，大所高所から知識なり情報を押しつけているとも考えられるでしょう．これこそLogoが最も忌み嫌うことであり，型にはまった画一的な人間を生みだす原因となっているのです．

Logoの理念では，子供はコンピュータに教わるのではなく，子供がコンピュータに教え込むことで，コンピュータ，つまりLogoと共に子供が学習してゆくことがベストであると考えます．ですから子供たちはLogoを使って何かをすることで，プログラムとはどんなものであるかを学ぶと共に，プログラムそのものの本質である，ものごとを考える手順であるとか，思考の方法，論理の立て方といった〝学習の方法〟を，自然に身につけてゆくことができるのです．子供の立場

から考えてみると，子供はLogoを通じてものごとの仕組みや世界の成立を，無理なく体得してゆくことになるのです．

タートル・グラフィックス

Logo＝タートル・グラフィックスと言われるほど有名なのが，Logo独得のタートル，つまりカメが図形を描いてゆくユニークなグラフィックス手法です．タートルという具体的な第三者が画面上に現れて，プログラムの結果を即座に実行（実演と言うべきか）するために，目的の図形を描かせるには次に何をさせたらよいか，がみずからのなかで判定できるため，プログラムという行為自体に親近感を抱けるようになります．また常に画面上で健気に動くタートルに対し感情移入が生まれ，Logoとの対話の中に暖かさが生じる効果も見逃せません（実際にこのカメさん，存外可愛いものです．BASICのエラーメッセージなんぞより，はるかに親近感を抱けますョ）．

コンピュータのおもしろさに単刀直入に没頭するには，やはりグラフィックスの世界が一番の早道でしょう．しかしこれまでのグラフィックスの手法，つまり画面をX軸とY軸に見立てて描画する方法では，プログラムの手順なり論理構造なりを覚えるまで，なまなかならぬ苦労を強いられたものでした（いったい，どうやって幼児に関数を教えたらよいものやら……）．タートル・グラフィックスでは，タートルに角度と距離を与えてやれば，少なくとも最小の要件は満たしたことになります．たとえばひし形を描いたり一点の角を中心に少しずつ図形を回転させるような場合，Logoと従来のグラフィックスとでは，その手順に雲泥の差がつきます．基本的なタートル・グラフィックスのコマンドは，実に簡単であり，命令と実行がひと目でわかるように作られているのです．だからタートルを動かして図形を描こうとする者は，プログラムの構造に対する理解の促進を，画面上を動くタートルを通じて得ることが可能となるわけです．

一般的に考えて，コンピュータでプログラミングを覚えようとする人が，最初につまずいてしまうのが，不毛とも思えるほど膨大な数の例題をこなすという，退屈にして面倒臭い関門をくぐらねばならないことでしょう（その行為に快感を覚える人ももちろんいるが……）．しかしLogoの場合，タートル・グラフィックスをまず最初に覚えてしまうと，自分で作成したプログラムに対し，変数，演算，条件判断，再帰呼出し，といったテクニックを次々と付加し，プログラム自体を拡張してゆくことができます．つまりLogoを学ぶ人の進歩に合わせ，プログラムも同時に進化してゆくというわけです．しかもそうした進化のプロセスで，プログラムの構造や組み立て方を，わかり易い目に見える形で提示して，教え学ばせてくれるのがタートル・グラフィックスの世界です．学ぶ側とコンピュータが，共に進歩してゆくというLogoの理念が，ここに生かされているのです．

(2) 進化した言語としてのLogo

Logoを用いたプログラミングでは，BASICとは異なった概念が導入されています．つまりLogoのプログラムは，全体がひとつの大きなプログラムとして成立しているのではなく，一つひとつを個別にモジュール化し，独立したプログラム（手続き）の組合せとして成立します．Logoではこのモジュールが次々とプログラムとして実行を続けることによって，目的の仕事を達成するわけですが，これをプログラムの構造化と呼んでいます．また，各々の独立したプログラム（モジュール）をプロシージャと呼び，名前を付けることが可能です．PascalやLISP，FORTRANといった言語でもこのような，プログラムを独立した手続きの集大成として構成することができますが，パソコンの標準言語たるBASICとなると，こんな芸当はとてもできません．BASICでは大きな問題，つまりプログラム全体を考えるために問題を細分化し，その後これを構成することが困難な構造になっています．ですからBASICで書かれた長いプログラムを見てゆくうちに，GOTO命令がリストのなかに大量に入っていたりすると，いったい何を実行するプログラムであるかがわからなくなったり，変数の設定が何を意味するのかが，さっぱり見当がつかないといったような事態に往々にして陥ります（ときとして自分で書いたプログラムでさえわかんなかったりして……）．プログラムをモジュール化できるというLogoの特性は，プログラムに実行させたい目的に従った全体の組み立てが容易にできるというメリットを生み，この結果デバッグの際の手順が大幅に簡略化されるという効果も生まれてくるわけです．

またLogoの特徴として，プロシージャのなかで自分自身（つまり同じ名前のプロシージャですな）を呼び出して実行させる，リカーシブコール（再帰呼出し）という機能が挙げられます．この機能は再帰的と呼ばれる特定のアルゴリズム（たとえば数学で教わる漸化式で表せるようなもの）をプログラミングする場合にたいへん役に立つもので，プログラムの構造をより簡単・明瞭にします．

(3) 人工知能型言語としてのLogo

人工知能型言語として名高いLISPの機能——リスト処理——を，Logoはずい分受け継いでいます．人工知能に欠かせない機能が言葉，すなわち文字の処理能力です．この文字を処理する基本が，リスト処理なのです．リスト処理についての詳細は4章で述べることにして，ここではその基本を簡単に説明しておきましょう．

まずは，ワードという言葉から．文字がいくつかつながったものをワードと呼びます．たとえば，

```
abc
A12CD
143
```

などはワードです．そして，ワードをいくつか並べて，［ ］（ブラケット）でくくったものがリ

ストです．たとえば，

[A B C]
[Sony SMC]

などです．さらにいくつかのリストをまとめて新たにリストとして扱うこともでき，

[[CBS Sony] SEIKO]
[MIKA [[MAMI] SAEKO]]

というのもリストになります．

こういったリストをいろいろと操るのがリスト処理なわけで，この説明だけではよくわからないでしょうが，これで人工知能っぽいことができるわけです．

Logoのこうした側面は，教育用言語としてのことば遊び等の利用法の他に，情報検索であるとかデータベースの構築などの，より高度な応用に対して新しい展望を与えることになります．

(4) Logoの位置付け

さて，いろいろとLogoの効能書きを並べたててきましたが，ここらでひとつ，結論めいたものを出しておきましょう．それは，Logoという言語は実に使い勝手がよい(最近のコトバで言えば，ユーザーフレンドリー）ということです．たとえばエラーメッセージの出し方ひとつにもそうした片鱗がうかがえて，

I don't know how to abc　(abcなんてプロシージャは，定義されてないから知らないよ)
Not enough inputs to box in box:　(boxのプロシージャに渡す引数が足りないぞ)

といった具合に，エラーを具体的に指摘してくれるのです．このようなエラーメッセージひとつを見ても，Logoの開発理念にこめられた，〝学習とは答えや答えを得る方法を入手するのではなく，間違いを検証し，プロセスを体得することで考える方法を手に入れるもの〟といった姿勢が見られます．しかもLogoは幼児でも使うことのできる言語ではありますが，その内に秘めたパフォーマンスは非常に高く，あらゆる世代の人が，あらゆる目的で活用することのできる言語でもあります．Logoが使う人と共に進歩してゆく賢い言語だということを念頭におき，積極的に活用して欲しいものです．

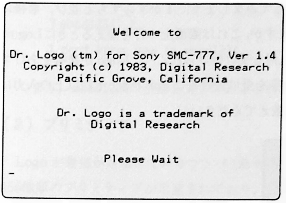
なお，Logoの理念をより詳しく知りたい方には，セイモア・パパートの著作『マインド・ストーム』の一読をお勧めします．

2-2 Logoに触れる前に

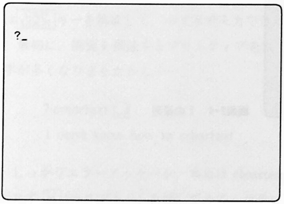
実際にLogoに触れる前に，Logoについて基本的なことがらを述べておきましょう．退屈だという方もいらっしゃるでしょうが，次のステップへの基礎力養成という意味で，ここは少しの間辛抱して読破してください．

(1) プロンプト状態

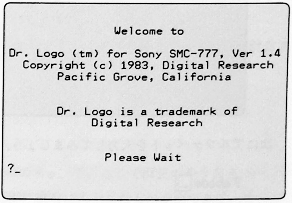
LogoがSony Filerのシステムメニューから呼び出されると，画面2－1の状態で少しの間止まり，次に画面2－2のように画面が消され，左上に？が表示され，_が点滅します．ただし，Logoとファイルを同時に呼び出すと，画面2－3のような画面となります．



画面2-1　Dr. Logo初期Wait画面



画面2-2　テキスト初期画面



画面2-3　初期プロンプト

この？をプロンプトと呼び，_の方はカーソルと呼びます．このプロンプトが表示されている状態を通常のプロンプト状態（そうでないのもある）と呼びます．このときLogoはキーボードからの命令の入力を受け付け，何か入力してやるとそれを実行し，また通常のプロンプト状態に戻ることになります．ただしこのときに入力したコトバがLogoに理解不能であった場合は，わからん，とエラーメッセージを表示して通常のプロンプト状態に戻ります（以下単にプロンプト状態と書

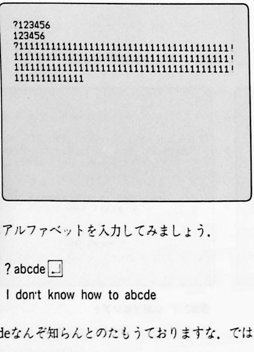
いたら，通常のプロンプト状態を指します)．まァ，とにかく何か入力してみましょう．

?123456………

デタラメにいくら数字を入力しても，カーソルは点滅したまま——つまり，Logoがこの数字の羅列をどうしたものか，判断に苦しんでいるのです．では，万国共通対コンピュータ命令手順であるところの[↵]を押してみましょう．

?123456[↵]
123456
?_

と表示されます．どーすりゃいいんだ，とプロンプトとカーソルが開いている状態です．しからば今度は1のキーから指を離さず，押しっ放しにしてみましょう．1がずらずらと並び，最後に音が出ます．画面上の各行の終りには！が出ていますが，これは画面上の行を変えるときにLogoが出すサインです．そして最後に出る音は，これ以上何も受け付けられん，とLogoが悲鳴をあげているのです．つまりLogoでは？の後に最大129文字を受け付けることができ，それ以上の入力はできないことを意味します（暇な方は1の数字を数えてください)．



画面2-4　1の羅列

次にアルファベットを入力してみましょう．

?abcde[↵]
I don't know how to abcde

abcdeなんぞ知らんとのたもうておりますな．では，

?konnichiwa[↵]
I don't know how to konnichiwa

あいさつも知らんと抜かしよります．これは別にLogoが礼儀知らずというわけではなく（もっとも，教えてやれば礼儀も覚えるのだから，ある意味じゃ礼儀知らずかも)，前述したとおり，理

解できるコトバ（命令）のみ実行してプロンプト状態に戻り，理解不能な場合はエラーメッセージを出してプロンプト状態に戻るという，Logoの規則を実行しているだけなのです．

しかし，最前に試した数字のときは，Logoはエラーメッセージなど出しませんでした．このことから類推すると，Logoは数字というコトバは知っているものの，文字の方は明確な命令になっていなければ受け付けてくれない，ということになります．実験してみましょう．

```
? 1234abcd [↵]
1234
I don't know how to abcd
```

やはり，数字はわかるけれど，abcdだけでは文字の方は理解できないと申しております．つまり，Logoでは命令を受けるときは，文字でなくては受け付けないということです．ちなみに，

```
? abcd1234 [↵]
I don't know how to abcd1234
```

つまり，abcd1234の全部を文字であると判断してしまうことを意味します．

(2) プリミティブ

Logoが最初から知っているコトバ（命令語）をプリミティブと呼びます．Dr. Logoには合計156種類のプリミティブが用意されており，これを組み合わせることで，Logoに様々な芸当を実行させることができるのです．このプリミティブ，Dr. Logoでは全部小文字で表現されます．まずは[CAP LOCK]キーを解除して，小文字で入力できるようにします．

最初に，画面を掃除するプリミティブを試してみましょう．イタズラ書きで，ずい分余計な文字が多くなりましたから……．

```
? creartext [↵]
I don't know how to creartext
```

しっかりエラーメッセージ．本当はcleartextなのですョ．で，ここで訂正のやり方を……．まず[H]（ヘルプキー）を押してみてください．画面に直前に入力したものが出てきて，カーソルがその右端にいます．

```
? creartext_
```

そしてカーソルを左へ動かしてcの隣のrのところへ持ってゆき[L]を押してください．

```
? clreartext
```

Logoでは常にカーソルの左に文字が挿入されるようになっていて，このようになります．カー

ソルはまだrのところにありますから、DELでこのrを消し、⏎を押してみましょう。

?cleartext⏎

みごとに画面は初期状態に戻りました。

ところでこのcleartextをctと省略しても同じ結果が得られます。文字数の多いプリミティブの多くには、このような短縮形が用意されているのです。

さて、もう少しプリミティブと付き合ってみることにしましょう。

?print [スペース] [Konnichiwa]⏎
Konnichiwa

ようやく礼儀を覚えてくれました。printというのはその次にある［ ］（ブラケット）で囲まれた中身を表示せよとのプリミティブ。ですからブラケットの中が大文字であろうがカタカナであろうが、一切気にせず表示します。しかもこのprintにはprという短縮形が用意されていますから、

?pr [コンニチワ]⏎
コンニチワ

やっと日本語であいさつするようになりました。

ところで、プリミティブとブラケットの間にスペースを置きましたが、これはLogoがプリミティブであることを理解するための約束事で、あらゆる場面に適用されます。以下プログラム中 [illegible]

[illegible]

（3）表示画面

cleartextを実行すると，画面はプロンプトとカーソルだけ，この画面の状態をテキスト表示画面と呼びます．Logoはこのテキストスクリーンの他に，グラフィックを表示するグラフィック表示画面を持っており，さらに双方を合わせたようなスプリットスクリーンもあります．それではこの3つの画面状態について，順を追って説明してゆくことにしましょう．

テキストスクリーン

テキストスクリーンの1行は40文字，ただし40文字目には前述の！マークが表示されるため，実質上は39文字となります．

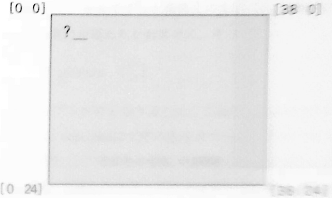


図2-1　テキストスクリーンの初期画面と座標

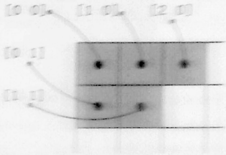


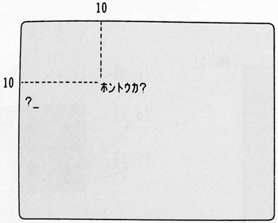
図2-2

```
? ct
? cursor
[0 1]
```

?setcursor [10 10]↵

プロンプトとカーソル，あっという間にすっ飛んでゆきます．ただしよく注意して数えてみると，プロンプトとカーソルは上から12行目，番地でいうと11行に出ていることがわかります．これは [10 10] の座標にカーソルが飛び，次の瞬間に改行してしまったことを意味します．本当かどうか確認してみましょう．

?ct setcursor [10 10] pr [ホントウカ？]↵



画面2-5　ホントウカ？

ホントウカ？のホの字が [10 10] の位置にあることがわかります．つまりカーソルが [10 10] の位置に飛んでprint命令を実行し，その後プロンプト状態に戻ったという証拠です．なお，このようにいくつもの命令をスペースで区切って一度に入力できることも覚えておいてください．

ところで，せっかくprint命令が再登場したのですから，これにちなんでテキストスクリーンに関する大切なプリミティブを紹介しておきましょう．

?type [ワイワイ] type [ガヤガヤ]↵
ワイワイガヤガヤ

これをprint命令で試してみると，

?print [ワイワイ] print [ガヤガヤ]↵
ワイワイ
ガヤガヤ

えらく違います. printは文字を表示したあとに改行するのに対して, typeは改行しない命令なのです.

このような print, type という命令と似たプリミティブに, show というものがあります.

```
?show [ハラヘッタ]⏎
[ハラヘッタ]
```

つまり, ブラケットまで表示してしまうのがshowなのです. なお, プロンプト状態であれば,

```
?[ハラヘッタ]
[ハラヘッタ]
```

というように, showを自動的に補って表示してくれます.

ところで, 今までずっと画面上の文字の色は白だけでした. 文字に色も付けられないのでは, Logoの実力が疑われかねません. そこで,

```
? setalpha 4 ⏎
```

プロンプトが赤くなりました. これで, 以降の文字は全部赤で表示されます. すでにおわかりのように, setalphaは文字の色をカラーコード4, つまり赤に指定するために使ったプリミティブです. また, この文字色の指定のとき, 8～15の値を入れてみると, 指定値－8の番号の色となり (setalphaの場合のカラーコードは0～7のみ), 該当の色のついた文字が点滅を始めます.

カラーコード	色	カラーコード	色
0	黒	8	黒 (点滅)
1	青	9	青 (〃)
2	緑	10	緑 (〃)
3	空	11	空 (〃)
4	赤	12	赤 (〃)
5	赤紫	13	赤紫 (〃)
6	黄	14	黄 (〃)
7	白	15	白 (〃)

表2-1 テキストスクリーンのカラーコード表

スプリットスクリーン

文字ばかり見ていてもおもしろくないので，ここらで有名なタートルに登場してもらいましょう．F2を押してください．すると画面2-6のようになりますが，真中に出現した矢印めいたものがタートルで，カーソルは画面の下から5行目のところに表示されています．これはテキストとグラフィックスが重なっている状態で，スプリットスクリーンと呼ばれます．この状態でのテキストは下の5行分だけですが，もちろん変更は可能です．

```
? setsplit 10↵
```

テキストは10行分に拡大されました．つまりsetsplitでテキストの大きさを変更できるということですが，このことはのちほど詳しく説明します．また，テキストスクリーン状態に戻したいときはF1を押します．

フルスクリーン

画面全体をグラフィック表示のみに使う画面設定で，F3を押すことで，この状態になります．文字やカーソルは表示されませんがキーボードからの入力は受け付けています．試しに何かデタラメを入力してみてください．エラーメッセージを表示するためスプリットスクリーンになってしまいますが，とにかく入力されたことはわかります．

ところでファンクションキーを使わず，プリミティブで画面状態を変更することもできます．文字どおり，

```
textscreen
splitscreen
fullscreen
```

の3つで，それぞれテキストスクリーン，スプリットスクリーン，フルスクリーンの状態に画面を設定します．

論理構造を持つ言語としてのLogoの成立を，本当はここいらで説明しておかなければならないのかもしれません．しかし，そんな七面倒臭い話より，やはりLogoは使って楽しいコトバである，と体感することの方が大事だと考えます(Logoの開発思想も，きっとコムズカしい事を悩みつつ学習せいとは言っとらんでしょう，きっと)．ですからここは独断で，まずLogoの楽しさを味わってもらい，しかるのち(もしくはそのプロセスで)Logoの構造なり論理性なりを把握してゆくようにしたいと思います．で，次にいよいよ，Logo入門の意味も含め，タートル・グラフィックスを試してみることにします．

2-3　タートル・グラフィックス入門

ようやく，実際にLogoを使うところにたどり着きました．タートル・グラフィックスは前述したように，従来のグラフィックスとは違い，タートルという明確なキャラクタ（と言ってよいものかどうか……)が，画面上を走り回って描画することがなによりの特徴となっています．BASIC等によるグラフィックスと比べて手順自体が格段に易しいうえ，タートルが描いてゆくということで，完成に至るプロセスを目で確かめられるため，グラフィックスに対する習熟の向上が望めます．そして，なにより有難いのは，タートル・グラフィックスを試みることで，Logoにおいて要求される手順のあらかたを学んでしまうことができるのです．まずは最も簡単なグラフィックスから――そのためには，タートルを呼び出さねばなりません．画面に向って，

「カメ，出てこい」

と声をかけつつ，[F2]を押してください．タートル・グラフィックスの始まりです．

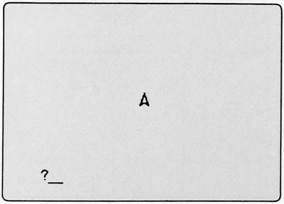
(1) カメ，動け！

前進・後退

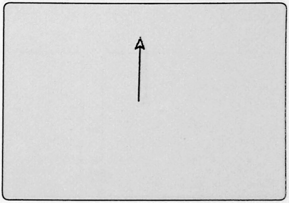
タートルをよく見てください．鋭った部分が前（つまり頭）で，前進するのはこの方向です．このタートルを動かすオマジナイであるプリミティブはfd（forwardの短縮形）でありまして，このfdに加えてタートルが何歩進んだものかを引数の値として与えてやると，前進を始めるのです．それではいよいよ，

?fd 50 [↵]

カメさん，歩いてゆきましたか？このfd 50という形式について考察を試みますと，Logoがfdなるプリミティブをスペースとスペースの間で読み取り，かつ次の50をプリミティブの引数として解釈したことでのタートル50ステップ分の前進，と定義できます．ですからこのとき，



画面2-6　スプリットスクリーン初期画面



fd 50

?fd50 ⏎

I don't know how to fd50

になるのは，fd50というプロシージャが定義されておらず，しかもプリミティブにfd50が存在しないことゆえのI don't knowであることがわかります．

では，次のプリミティブ．進む一方ではレミングの行進と変りませんので，ひとつバックさせてもとの位置に戻してみましょう．50歩前進したのですから50歩分後退すればよいわけですから，

?bk 50 ⏎

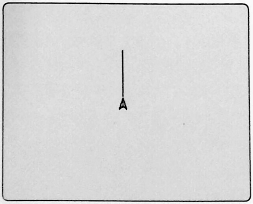
もとのサヤに収まりましたね．このbk，御想像のとおりbackの短縮形(もちろんbackやforwardでもそのまま使えます)．タートルが前を向いたまま後退したことで，本当にバックしていることがわかります．

ここで気が付くのは，タートルはもとの位置に戻っても，描いた線はそのまま残っているということです．これはLogoが起動したときにタートルの持つ描画力が規定され，常に線を引いている状態に設定されていることを意味します．つまりfd 50で直線を引いたタートルは，帰り道のbk 50の途上でも，直線を引きながら戻ってきたというわけで，この状態をpendown (pd) と呼びます．

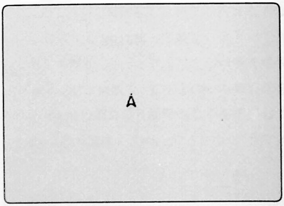
タートルを次の仕事に移らせたいのですが，どうも以前に描いた50歩分の線が邪魔．そこで，

?clean ⏎

このプリミティブは，タートルの位置や向きはそのままで，グラフィック表示のみを消す働きをするものです．cleanと似たプリミティブにclearscreen (cs) というものがあります．cleanとcsの違いは，前者がグラフィック表示を消すだけなのに対し，後者はタートルを初期の位置，つまり真中に戻す働きも併せ持っていることです．このタートルの初期位置である画面中央をho



?fd 50
?bk 50



?fd 50
?bk 50
?clean

me位置と言い，タートルを単にそこに戻すには，homeというプリミティブを使います．では，この3者の違いを実際に見てみましょう．

? fd 50 ↵
? clean ↵

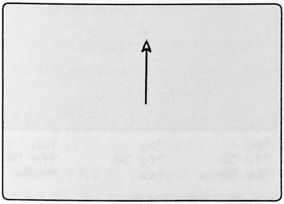
タートルは画面中央から50歩の位置にいて，描線は消されています．次に，

? fd 50 ↵
? home ↵

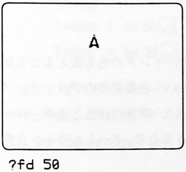
タートルは一度50歩さきまで進み，次いでhome位置に戻っては来ましたが，しかし描線はそのままです．homeというプリミティブに，グラフィック表示を消す機能がないことを意味します．それでは3番目，

? fd 50 ↵
? cs ↵

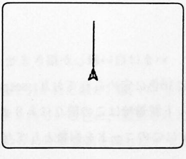
タートルは50歩進み，csと共にグラフィック表示の消滅に乗じてhome位置に戻りました．



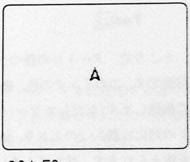
?fd 50



?fd 50
?clean



?fd 50
?home



?fd 50
?cs

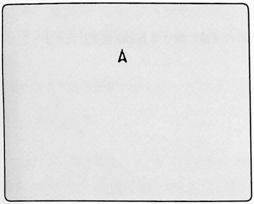
ペンについて

ところで，これまではタートルが線を引きっぱなしの状態，つまりpendownの場合に限って話を進めてきました．ここらで線を引かない場合，要するにペンを画面から離した状態であるpenup (pu) を試してみることにしましょう．

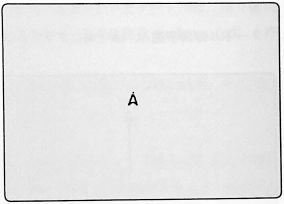
```
?pu ↵
?fd 50 ↵
```

タートルはペンを持ち上げたまま，線を引かずに50歩前に進みます．そして home 位置に戻すのですが，なにしろペンが上がったままですので，

```
?bk 50 ↵
?cs ↵
?home ↵
```



```
?pu
?fd 50
```



```
?pu
?fd 50
?bk 50
```

```
?pu
?fd 50
?cs
```

```
?pu
?fd 50
?home
```

のどれでも同じ結果，すなわち何をやっても home 位置に帰って来ます．タートルが戻ったなら，忘れずにペンを下ろさせておきましょう．

```
?pd ↵
```

ところで，タートルの持つペン，いまは白い線しか描きませんが，インクの色を変えることが可能です．このインクの色，通常は16色に定められており，setpc という色変更用のプリミティブで機能します(ただしカラーパレット装着時はこの限りにありません)．標準の16色とカラーコードの対応は表2－2のとおり，setpc にこのコードを引数として与えることで，タートルのインク色を変えられます．試してみましょう．

カラーコード	色	カラーコード	色
0	黒	8	暗緑
1	青	9	灰緑
2	緑	10	朱
3	空	11	山吹
4	赤	12	灰青
5	赤紫	13	淡青
6	黄	14	桃
7	白	15	灰

表2-2　ペンのカラーコード表

```
?setpc 1↵
?fd 50↵
```

青い線が50ステップ分引かれました．もしこのとき，カラーコード外の引数を与えると，

```
?setpc 16↵
setpc doesn't like 16 as input
```

つまり引数の値がプリミティブの要求する範囲に見合わない，とメッセージが返されます．また，現在用いているペンのカラーコードを知りたいときは，

```
?pc↵
1
```

と教えてくれます．このpcは，pencolorというプリミティブの短縮形．タートルのペン色に対応するカラーコードの値を返す働きを持っているのです．

次に，インクの色の調合を考えてみましょう．まず，

```
?px↵
?setpc 7 fd 50↵
?setpc 4 bk 50↵
```

白，つまりカラーコード7の色で引かれた線が，bkのときにカラーコード4に対応する赤で引かれません．これは，すでに表示されている色に，変更されたインク色が調合されたことを意味し，px（penreverse）は色を重ね合わせる機能を持ったプリミティブです．ところで，

```
?pen↵
```

と入力すると，

[PENREVERSE 4]

と表示されます．penは，現在のタートルの状態と色を返すプリミティブ．ここで大文字による表示が出てきましたが，これはLogo内部で使われているコトバのこと．penによるタートルの状態と色の表示をペン情報リストと呼びますが，setpenというプリミティブを使うことにより，この内容を一度に書き換えることもできます．

?setpen [PENDOWN 7] ⏎

これで，ペンの上げ下げもペン色も最初の設定に戻ります．setpenに与える引数はpenで得られる情報リストと同じ形式で，ペン状態を示すのには大文字が使われます．

なおペンの状態にはPENUP, PENDOWN, PENREVERSEのほかにPENERASEというものがあり，これは文字どおり，タートルが消しゴムを持って動くような状態です．penerase（短縮形はpe）というプリミティブでペンをこの状態にすることもできます．

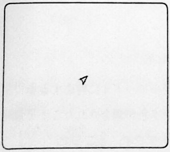
タートルの向きを変える

前後の描線の次は，方向を変える手段を考えてみましょう．まず，タートルの角度を変えてやらねばなりません．進行方向に対して左右方向に角度を変えるのですから，プリミティブはleftとright，それぞれltとrtに短縮できます．ではさっそく，

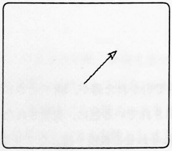
?rt 45 ⏎

カメ，頭を振りました．すでにおわかりのように，rtのあとに与えた45という引数はタートルの角度の値です．rtとltでは，進行方向に対する角度が引数となるのです．次に直進させてみましょう．

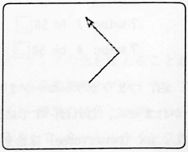
?fd 50 ⏎



?rt 45



?rt 45
?fd 50



?rt 45
?fd 50
?lt 90 fd 50

真上に対して45度の角度で50歩前進．これだけでは芸がないので今度はさらに左90度ひねりの50歩前進です．

?lt 90 fd 50 ↵

ところで，タートルは姿を消すこともできるのです．hideturtle なるプリミティブでありまして，短縮形は ht です．

?cs ht ↵

しかもこの状態でもタートルはしっかり働きます．おまけに姿を現しているときより，動きは速くなります．

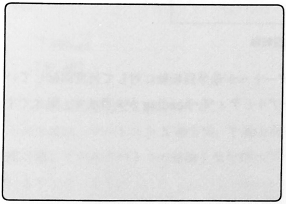
?rt 90 fd 50 lt 90 fd 50 ↵

これで線だけが描かれます．再びタートルを登場させるためのプリミティブは showturtle，短縮形は st です．

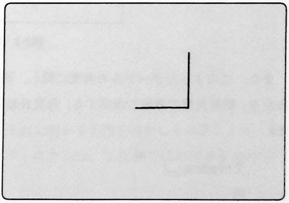
?cs st ↵

また，タートルが画面上に表示されているかどうかを確認するためのプリミティブも用意されています．これは shownp なるもので，st の状態であれば TRUE と答え，ht ならばFALSE とメッセージを返してよこします．

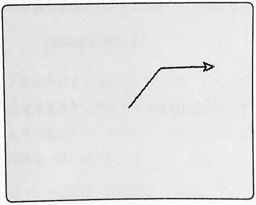
さて，タートルに角度を与えるプリミティブが lt と rt であることはわかりました．しかし，実はこの 2 つ以外にも角度に関するプリミティブがあるのです．rt と lt はタートルが進んでいる(頭が指している)方向に対して角度を与えるもので，これは相対角度を与えるプリミティブであると言えるでしょう．が，画面上における基準方向に基いた角度，つまり絶対角度に対しては rt も lt も無力です．タートルが初期設定である home 位置にあるとき絶対角度（基準回転軸に対す



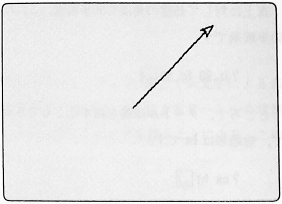
?cs ht



?rt 90 fd 50 lt 90 fd 50



?cs rt 45 fd 50 rt 45 fd 50



?cs seth 45 fd 50 seth 45 fd 50

る角度)は0度であり，これをもとに角度を決めてやるプリミティブが必要となります．で，その名前をsetheading，略称sethといいます．rt，ltとの相違を確かめてみましょう．

?cs rt 45 fd 50 rt 45 fd 50↵

?cs seth 45 fd 50 seth 45 fd 50↵

両者の描線を見れば一目瞭然．rtとltがタートルの進行方向に対して角度を与えるプリミティブであるのに対し，sethは基準回転軸からのみの角度を与えるプリミティブであることがわかります．

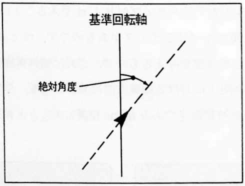


図2-3　基準回転軸

また，このようなタートルの角度に関し，現在タートルが基準回転軸に対して何度回転しているかを，絶対角度の数値で表示する，角度お知らせプリミティブ，headingがあります．現状ですと，

?heading↵

45

と，答えてくれる便利モノでありました．

(2) カメの縄張り

グラフィックス座標

タートルは，グラフィックスクリーン上において，移動できる範囲が定められています．home位置を基点として，

右へ　126歩
左へ　127歩
前へ　99歩
後へ　100歩

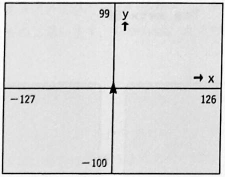


図2-4　タートル・グラフィックス座標

がタートルの活動エリアということになります．図2-4のように，これはＸ軸とＹ軸による座標と考えられるわけですから，逆にこれを利用してやりましょう．タートルに座標値（Ｘ軸，Ｙ軸での各座標値）を与えて動かしてやるのです．使用するプリミティブはsetpos（セットポジションの意味）．では試しに，

```
?setpos [90 40]⏎
```

タートルは，Ｘ軸方向に90歩，Ｙ軸方向に40歩移動します．別に角度を与えたわけではないので，タートルの向きは変りません．

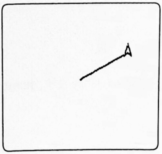
次にタートルの現在位置を知るプリミティブの紹介．このためにはpos（ポジションの意味）なるプリミティブを使います．タートルの居場所を座標値で知らせてくれる働きを持っており，今なら，

```
?pos⏎
[90 40]
```

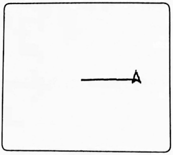
といった具合に答えてくれるのです．

では今度は，タートルをＸ軸方向，Ｙ軸方向それぞれに動かす手段を紹介してみましょう．Ｘ座標に関してのみタートルの移動を受け持つプリミティブをsetx，Ｙ座標だけの担当をsetyと言います．さっそく，

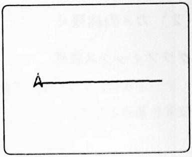
```
?cs setx 80⏎
```



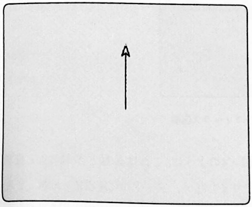
```
?setpos [90 40]
```



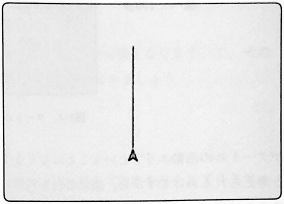
```
?cs setx 80
```



```
?cs setx 80
?setx -80
?xcor
-80
```



```
?cs sety 50
```



```
?cs sety 50
?sety -50
?ycor
-50
```

タートルは，角度を変えないまま，水平にｘ＝80の位置に移動しました．逆もやってみましょう．

?set x －80↵

やはり指定座標，ｘ＝－80に走って行きました．現在位置の確認は次のとおり．

?xcor↵（x コーディネイトの意味）

－80

X 座標だけでは片手落ち，おつぎはＹ座標．

?cs sety 50↵

ｙ＝50の位置に直進します（csを入れないとX軸が最前の－80のままです）．逆も試して，

?sety －50↵

きちんとy＝−50に動きました．確認は，

```
?ycor ↵
−50
```

カメの行動範囲

さて，タートルにはグラフィックスクリーン上の活動エリアが定められている，という認識は座標の話で御理解いただけたものと思います．しからば，カメの奴がこの縄張りを超えたらどうなるのか．試みに，

```
?fd 10000 ↵
```

画面の外にぶっ飛んで消えてしまいます(当たり前だ)．これでは始末に困るので，タートルの移動範囲を限定します．カメの遊び場を柵で囲うようなものでしょう．プリミティブはfenceです．

```
?cs fence ↵
```

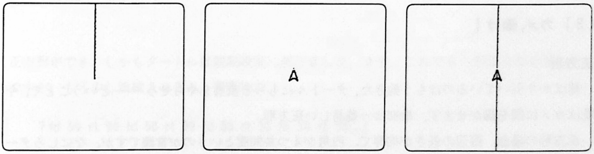
これでタートルの囲い込みが完了．では実験．

```
?fd 10000 ↵
Turtle out of bounds
```

タートル立入禁止区域に侵入計画中，てなメッセージが返されてきました．命令に縛られて，タートルの方は一歩も動けません．この囲いこみの状態をフェンス状態と呼びますが，もしタートルがすでに画面外に出てしまっている場合には，fenceは実行不可能です．

タートルの行動範囲を規制するもうひとつの方法は，wrapというプリミティブを使うもの．

```
?wrap fd 10000 ↵
```



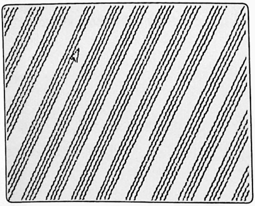
```
?cs
?fd 10000
```

```
?cs fence
?fd 10000
Turtle out of bounds
```

```
?wrap fd 10000
```

線は引いたものの，タートルは真中(home)に留まったまま．wrapによる行動規制下では，タートルが画面からはみ出すようなマネをした場合，画面の上下，左右が連続しているように，画面の反対側にタートルが移動するような状態にします．これをラップ状態と呼びますが，わかり易くするために，ちょっとタートルの角度を変えてみましょう．

?cs wrap rt 30 fd 10000 ⏎



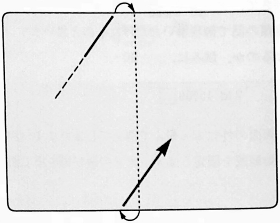


図2-5 ラップ状態

御理解いただけましたかな．タートルは画面を飛び出しては仮想のループを回って戻り，10000歩分を前進したわけです．

このように，フェンス状態，ラップ状態にすることでタートルの行動エリアを規制できるわけですが，これらの状態を解除してやるには，windowというプリミティブを使います．

?cs window ⏎

これで，タートルは再び首輪を外されたことは，あまりやりたくはない実験で証明されます．

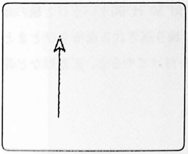
?fd 10000 ⏎

(3) カメ，働け！

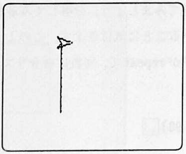
正方形

線ばかり引いているのはもう飽きた．タートルにもっと芸当をやらせろ —— ということで，今度はカメに図を描かせます．最初は一番易しい正方形．

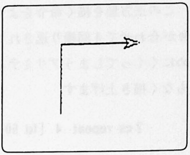
正方形の場合，四辺の長さが均等で，内角が4つ共90度というのが常識ですが，なにしろタートルが描いてゆくのですから，描く過程を決めてやらねばなりません．ここでは，一辺の長さが50歩で，かつタートルが右回転で描いてゆく正方形を試みます．



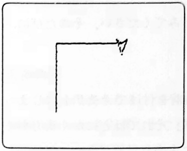
?fd 50



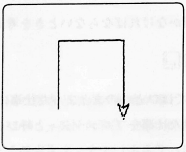
?fd 50
?rt 90



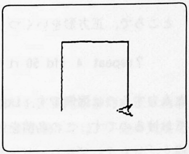
?fd 50
?rt 90
?fd 50



?fd 50
?rt 90
?fd 50
?rt 90

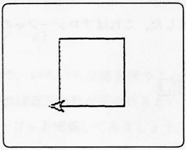


?rt 90
?fd 50
?rt 90
?fd 50

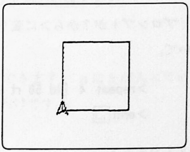


?fd 50
?rt 90
?fd 50
?rt 90

```
?fd 50  ↵
?rt 90  ↵
?fd 50  ↵
?rt 90  ↵
?fd 50  ↵
?rt 90  ↵
?fd 50  ↵
?rt 90  ↵
```



?rt 90
?fd 50
?rt 90
?fd 50



?fd 50
?rt 90
?fd 50
?rt 90

正方形ができ，しかもタートルは初期設定に戻りました．まァ，これでも一応正方形を描けるのですが，もう少し洗練させると，御存知の，

```
?fd 50 rt 90 fd 50 rt 90 fd 50 rt 90 fd 50 rt 90 ↵
```

となり，同じことが1行で済んでしまいます．しかし，これでもまだ，まどろっこしいとは思いませんか？

この正方形を描く命令をよく見てみましょう．分解してみると，fd 50 rt 90というひと組の命令が合わせて4回繰り返されていることに気付きます．このように繰り返される命令をひとまとめにくくってしまうプリミティブがrepeatで，回数に命令リストを付けてやると，正方形など苦もなく描き上げます．

```
?cs repeat 4 [fd 50 rt 90]↵
```

かなりシンプルになりました．もうおわかりのように，repeatは後の[ ]内の処理を繰り返し実行せよとの命令で，[ ]の前の数字は繰り返しの回数を指定しています．

ところで，正方形をいくつも描かなければならないときを考えてみてください．そのたびに，

```
?repeat 4 [fd 50 rt 90]↵
```

と入力するのは面倒です．Logoではひとつのまとまった仕事に名前を付けてメモリ上にしまっておけるのです．この名前を付けた仕事をプロシージャと呼びます．それではとにかく正方形を描く「シカク」プロシージャを作ってみましょう．まず次のように入力してください．

```
?to シカク↵
>
```

プロンプトが?から>に変りました．これはプロシージャの定義中であることを示します．続いて，

```
>repeat 4 [fd 50 rt 90]↵
>end↵
```

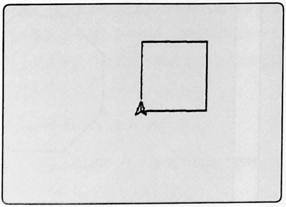
と入力すると，

```
シカク defined
```

と表示して通常のプロンプト[?]が出てきました．シカクdefined（シカクは定義されました）と言っていますから，ほんとうに定義されているかどうか調べてみましょう．プロシージャを呼び出すのはいたって簡単．その名前を入力すればよいのです．

```
?シカク↵
```

予定どおり正方形が描かれました．これ以降「シカク」と何回入力しても，同じように正方形が描かれます．



?シカク

正多角形

正方形に代表される，すべての辺の長さと頂点の角度が等しい図形を正多角形または正Ｎ角形と呼びます(今さら書くまでもないか…)．正Ｎ角形は一定の長さを fd で進んでは，一定の角度だけ曲がることをＮ回繰り返すことで描けるはずです．つまり，

```
repeat n [fd 一辺の長さ rt 360÷n]
```

ということになります．これを一辺の長さを50として，Logo のコトバで定義すると，

```
repeat :n [fd 50 rt 360 / :n]
```

となります．:n は変数というもので，いろいろな値を持つことができます．:n に 5 が入っていれば正五角形，:n に 3 が入っていれば正三角形が描かれるというわけです．

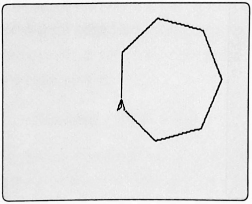
それでは「タカクケイ」プロシージャを定義してみましょう．

```
?to タカクケイ :n↵
>cs↵
>repeat :n [fd 50 rt 360 / :n]↵
>end↵
タカクケイ defined
```

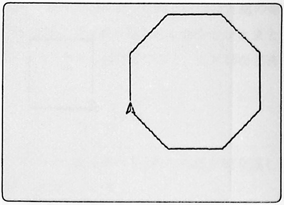
今度は to タカクケイのあとに :n が書いてあります．こうしておくと，

```
?タカクケイ 8↵
```

と入力すれば :n に 8 が代入され，正八角形が描けるわけです．fd とか rt とかいうプリミティブと同じように，プロシージャにも引数を持ったものが作れるのです．



?ﾀｶｸｹｲ 7

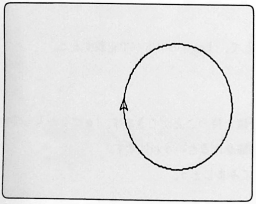


?ﾀｶｸｹｲ 8

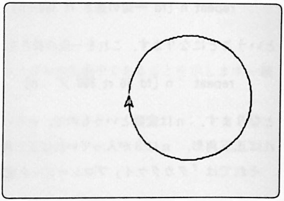
円

正多角形の頂点を無限に多くすると円になります．無限といわずとも，

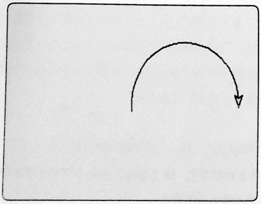
? repeat 360 [fd 1 rt 1]⏎



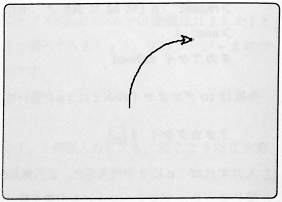
?repeat 360 [fd 1 rt 1]



?repeat 36 [fd 10 rt 10]



?repeat 18 [fd 10 rt 10]



?repeat 9 [fd 10 rt 10]

で何ら問題はありません. これでタートルが何をするのかといえば, 一歩前進するごとに右へ1度ずつ向きを変えてゆき, その作業を360回繰り返すことで円を描くというわけです. まァ, 確かに円にはなりますが, これではノロ臭くてアクビが出ます. ここは考え方を変え, 正36角形を描かせることにしましょう.

```
?repeat 36 [fd 10 rt 10]↵
```

さきほどの円とほぼ同じものを描くのに, 格段のスピードアップが達成されました. で, この考え方を応用すると, 半円であれば,

```
?repeat 18 [fd 10 rt 10]↵
```

になりますし, 4分の1だけ円を描くには,

```
?repeat 9 [fd 10 rt 10]↵
```

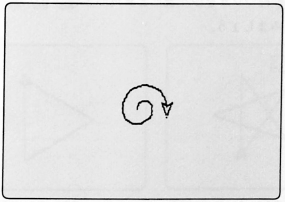
と表現することが可能です.

ところでこういった半円を直径を次々と広げて描いていったらどうなるでしょうか? 渦巻きになるはずです. では直径の変更可能な半円を作るプロシージャを作ってみましょう.

```
?to ハンエン :size↵
>repeat 18 [fd :size rt 10]↵
>end↵
ハンエン defined
```

このようにプロシージャが定義できたら,

```
? ハンエン 1 ハンエン 2 ハンエン 3…………
```



?ハンエン 1 ハンエン 2 ハンエン 3

と入力．確かに渦巻きはできますが，はっきり言ってまだるっこい．洗練させましょう．何か変数を用意して，それを1ずつ増やしながら，「ハンエン」プロシージャにそれを渡せばよいようです．

```
?to ウズマキ⏎
>make "r 1⏎
>reapeat 16 [ハンエン :r make "r :r + 1]⏎
>end⏎
ウズマキ defined
```

こうすると，目的どおりのプロシージャになります．ここでmakeというのは，変数に値を代入するプリミティブで，

make "〈変数名〉〈値〉

という書式で使います．repeat のなかで使われている，

make "r :r + 1

をこの書式と見比べると，r という変数名を持つ変数に，:r＋1を代入していることがわかるでしょう．

この渦巻きプロシージャでは，前に作った"ハンエン"のプロシージャが使われています．これはLogoでは，プロシージャも一度定義してしまえば，以後(電源を切るまでは)プリミティブと同じように使えるという特性ゆえなのです．

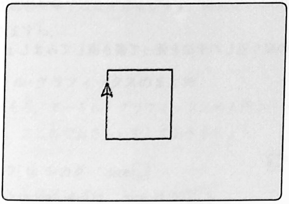
繰り返しと停止

さて，repeatで多角形を描いたときは，角数nのみを変数として取り扱い，辺の長さは50ステップに固定したままでした．しかしLogoの持つワザを少々使うと多角形を別のやり方で描くことが可能です．まずプロシージャを作ってみましょう．

```
?to タカクケイ :ヘン :カクド⏎
>fd :ヘン⏎
>rt :カクド⏎
>タカクケイ :ヘン :カクド⏎
> end⏎
タカクケイ defined
```

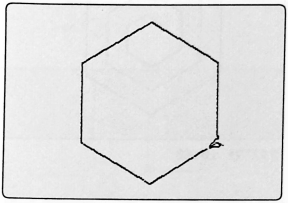
試みに，90度角で正方形を描かせてみましょう．辺の長さは60として，

? タカクケイ 60 90⏎

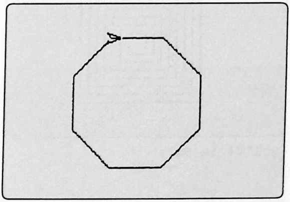


?ﾀｶｸｹｲ 60 90

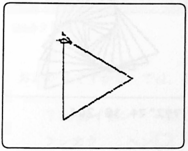
タートルが同じ進路，つまり60ステップ90度角の正方形を何回も描きます．



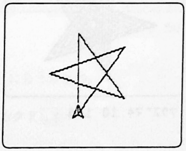
?ﾀｶｸｹｲ 60 60



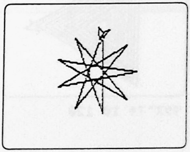
?ﾀｶｸｹｲ 40 45



?ﾀｶｸｹｲ 60 120



?ﾀｶｸｹｲ 60 144

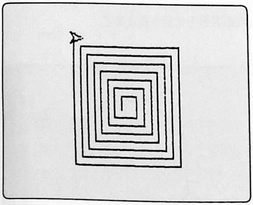


?ﾀｶｸｹｲ 60 200

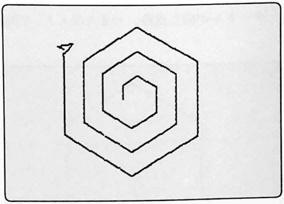
このプロシージャをよく見ると"タカクケイ"というプロシージャのなかで，さらに自分自身である"タカクケイ"を呼んでいることがわかります．Logoではこのように，任意のプロシージャから自分自身を呼び出すことができるのです．ただ前述のrepeatと違い，繰り返す回数を指定しないので，何回でも繰り返しているのです．このプロシージャの実行停止の際は，[CTRL]と[ESC]の2つのキーを同時に押します．

さきほどの渦巻きのプロシージャを，この繰り返しの手法を使って書き直してみましょう．

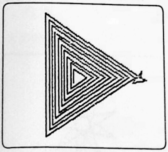
```
? to ウズマキ :ヘン :カクド ↵
>fd :ヘン ↵
>rt :カクド ↵
>ウズマキ :ヘン + 3 :カクド ↵
>end ↵
ウズマキ defind
```



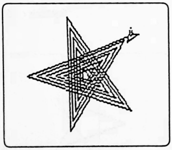
?ｳｽﾞﾏｷ 10 90



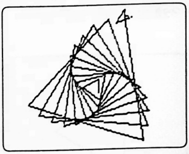
?ｳｽﾞﾏｷ 10 60



?ｳｽﾞﾏｷ 10 120



?ｳｽﾞﾏｷ 10 144



?ｳｽﾞﾏｷ 10 244

ここでもまた，変数に任意の値を代入することで，さまざまな渦巻きが出現します．この渦巻きプロシージャが多角形と異なるのは，プロシージャ自身がウズマキを呼び出したときに，：ヘンが：ヘン＋3になっていることで，各辺が前のものより3ステップずつ長くなり，以降これを延々と繰り返してゆくことを意味します．簡単なプロシージャでも思いのほか複雑な図形を描くことができますね．

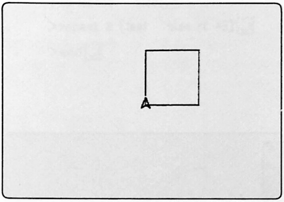
タートル・グラフィックスのまとめ

そろそろ，タートル・グラフィックスの入門コースを修了できるあたりまで漕ぎつけました．では，ここらでおさらいをしておきましょう．

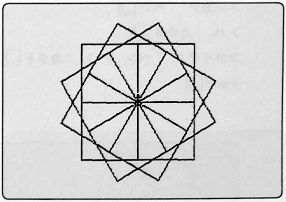
```
? to シカク :size⏎
>repeat 4 [fd :size rt 90]⏎
>end⏎
```

ここまでは，よくわかっているものと思いますので，次には，

```
?to ウゴク :size⏎
>repeat 12 [シカク :size rt 30]⏎
>end⏎
```



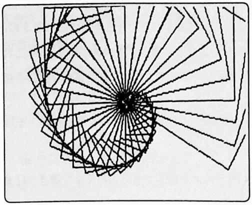
?ｼｶｸ 50



?ｳｺﾞｸ 50

おわかりですかな．では，さらにもうひとつ，

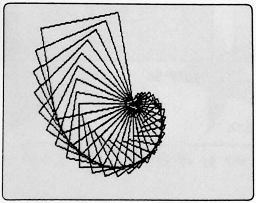
```
?to カイテン :ヘン :カクド⏎
>シカク :ヘン⏎
>rt :カクド⏎
>カイテン :ヘン + 3 :カクド⏎
>end ⏎
```



カイテン 1 10

さあどうです．御理解いただけましたか．一辺がNステップの大きさの四角形から始まって，次には3ステップずつ大きくなると共に，カクドで指定した角度ずつ回転してゆくことをいつまでも繰り返すプロシージャです．このようなプロシージャは留まるところを知らぬかのように実行を続けてゆくものですが，stopなるプリミティブを使うことで，上限を定めることも可能です．プロシージャを打ち直して，STOPをかけてみましょう．

```
?to カイテン :ヘン :カクド↵
>if :ヘン > 80 [stop] ↵
>シカク :ヘン↵
>rt :カクド↵
>カイテン :ヘン + 3 :カクド↵
>end↵
```



?カイテン 1 10　　　　（[STOP]付き）

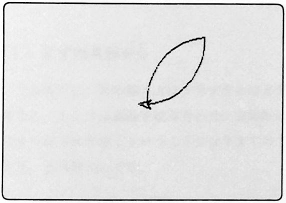
アンダーラインの行が停止規則を意味し，1辺が80ステップに達したらストップせよという命令になります．ここで使われているifというプリミティブは，条件判断のときに使われるもので，特定の状況ではこうなり，別の状況ではああなる，といったように与えられた条件によって実行内容を変えてゆくものです．ifのあとには判定文と命令リストが必要で，判定文によって設定される条件を，命令で実行させせることになります．ですからこの場合は，：ヘンが80ステップより大きくなった場合，判定文によって停止が実行されるわけです．

さて，タートル・グラフィックス入門も，そろそろ終りとなります．簡単に作れるプロシージャでも，ずい分と複雑な図形が描けると思いませんか．あとは，今まで学んだものを，自分なりに組み立ててオリジナリティを加えること．より複雑なグラフィックスは次章にゆずりますが，おさらいの意味をこめ，ひとつサンプルを書いておきましょう．

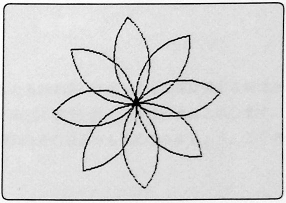
```
?to leaf :size ↵
>setpc 10 ↵
>repeat 9 [fd :size rt 10] ↵
>rt 90 ↵
>repeat 9 [fd :size rt 10] ↵
>end ↵
```

これを使って，

```
?to flower :size ↵
>repeat 8 [leaf :size rt 45] ↵
>end ↵
```



?leaf 10



?flower 10

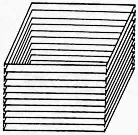


図3-1　直方体

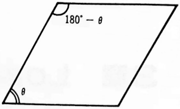


図3-2　角度θの平行四辺形

これで任意の大きさの正方形が得られるわけですが，立体らしく見せるには，底面は正方形ではなく，むしろ平行四辺形の方が似つかわしい．そこで図３－２のような，角度θを持つ平行四辺形を作ります．リスト３－１がそのプロシージャなのですが，話を先に進める前に，ここらでエディットモードに触れておきましょう．合理的なプログラミングには，なくてはならない知識なのです．

List 3-1

```
to ﾃｲﾒﾝ :l :a
rt 90 - :a
repeat 2
 [fd :l rt :a
  fd :l rt 180 - :a
 ]
 lt 90 - :a
 end
```

(2) プロシージャはエディットモードで

これまでは，プロシージャの定義は to～をプロンプト状態から入力することで行ってきました．しかし，この方法では↵を押してしまったなら，入力したものを訂正することは一切できません．ドジを踏んだら最初からやり直さねばならないわけですが，これではあまりに不便．そこで使われるのが，エディットモードで，プリミティブは edit (ed)です．

?ed↵

と入力することで，画面がテキストスクリーン状態となります（このとき？は表示されずカーソルのみが点滅します)．そこで最初の行に，

to ──↵

と入力し，それ以降に定義内容を続けてやり，最終行に，

end

を書いて F5 を押せば，エディットモードから抜け出して，同時にプロシージャは定義されてしまいます．

```
. edを実行するとカーソルが画面左上に現れる
to ﾃｲﾒﾝ :l :a↵
rt 90 - :a↵
repeat 2↵
 [fd :l rt :a↵
  fd :l rt 180 - :a↵
 ]↵
lt 90 - :a↵
end F5 ………endのあとは F5 を押す
```

edの実行例(登録)

エディットモードではカーソルパドルでカーソルを自由に動かすことができ，またキーボードからの入力は，カーソルのすぐ左に挿入されます．なお，エディットモード中における特殊なキーの機能を次のページの表3-1に示しますので，参考にしてください．

F5 を押して定義してしまったものを修正するには，

?ed "プロシージャ名↵

とすることで，再びエディットモードを呼び出せます．このときに忘れてはならないのは(makeのように)，" です．またエディットモード中にプロシージャ名を変えただけで F5 を押すと，そのプロシージャの名前が変るのではなく，新しいプロシージャが作られてしまいますので，注意した方がよいでしょう．

このほかのエディットモード関係のプリミティブとしては，まず現有のものすべてのプロシージャをエディットするためのedallがありますが，なにしろDr. Logoで一度にエディットできる量の限界が低いものですから，あまり使わない方が賢明．エディットする量が多くなり，処理の限界を超えてしまうと，

My edit buffer is full

とメッセージを返してきます．やはりおとなしくedでエディットするのが一般的と言えるでしょう．

キー	単独押しの場合	SHIFT と同時押し
RUB OUT BS	カーソルのある位置の１つ前の文字を消す カーソル以降の文字は１字分ずつ前へ動く	
→	カーソルが１つ右へ動く	カーソルが，画面上でカーソルのある行の最後へ動く
↑	カーソルが１つ上へ動く	エディットする内容が数画面に渡るとき，前の画面に戻る
↓	カーソルが１つ下へ動く	エディットする内容が数画面に渡るとき，次の画面に移る
←	カーソルが１つ左へ動く	カーソルが，画面上の行の先頭へ動く
HOME	カーソルがエディットすべき内容の最初の文字に戻る	カーソルが エディットすべき内容の最後の文字の次に移動する
CLR	カーソルのある位置の文字から，プログラムとしての１行の行末までを消す	無効
DEL	カーソルのある位置の文字を消し，それ以降の文字を１字分前へ詰める	無効
INS	カーソルのある位置に１行分のブランクを挿入する．カーソルがあった位置以降の文字は，１行ずつ下へ移動し，画面の最下行にあった行は消える	無効
TAB	次のタブ位置まで空白を挿入	無効
H	CLR で消したものが出てくる	無効

表3-1 エディットモードにおける特殊なキーの機能

なお，プログラムがエラーで停止したときに，ただ単に ed と入力すると，カーソルがエラーの発生したプロシージャのエラー発生箇所で点滅しますので，デバッグ時に便利だということも書き加えておきましょう．

次に，現在定義されているすべてのプロシージャを調べるためには，

? pots ↵

と入力します．これで全部のプロシージャ名が表示されますので，たとえば前に付けた名前を忘れてプロシージャを実行できないようなとき，とても重宝するものです．

また，特定のプロシージャの内容を見たいときには，

po "プロシージャ名

でOK．poallを使えば，ワークスペース内の全部のプロシージャの内容が，あとで触れるグローバル変数まで含めて表示されます．すでにおわかりと思いますが，これ以降の主なプロシージャのリストは，プロンプト状態からではなく，エディットモードで入力する形で載せますので，覚えておいてください．

(3) 底面そして高さ

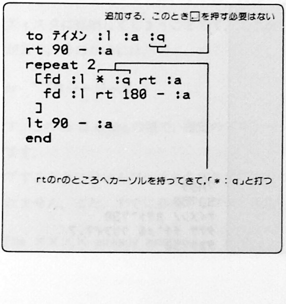
リスト3-1のプロシージャ，テイメンを用いて，とにかく底面を作ってみましょう．この長さは50で，角 θ は30°とします．

? cs テイメン 50 30 ⏎

これでひし形ができました．しかし，これでは奥行きがあり過ぎで，底面らしく見せるにはこれを短縮する必要があります．そこで，奥行きの長さの縮小率を指定できるようにプロシージャを変更し，リスト3-2のようなものを作ります．

List 3-2

```
to テイメン :l :a :q
rt 90 - :a
repeat 2
 [fd :l * :q rt :a
  fd :l rt 180 - :a
 ]
lt 90 - :a
end
```



edの実行例（修正）

リスト3－1との違いは，2つだけ．

?ed "テイメン ⏎

でエディットモードに入り，これらを追加・訂正したら F5 を押して実行します．

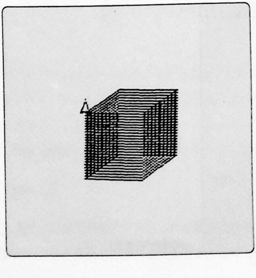
?cs テイメン 50 30　0.3 ⏎

これで，底面はやっとそれらしくなりました．しかし各辺の色が同じでは Logo のせっかくのカラフルな機能を無視することになってしまいます．そこでリスト3－3のように手を加えて，よりプログラムっぽく（?）見せるようにしてみました．

List 3-3

```
to テイメン :l :a :q
rt 90 - :a
setpc 13
fd :l * :q rt :a
setpc 12
fd :l rt 180 - :a
setpc 1
fd :l * :q rt :a
setpc 3
fd :l rt 180 - :a
lt 90 - :a
end
```

さて，こうして作られた底面も，正しく見ればただの線描の平行四辺形．こいつを使って直方体を作るためには，いくつも積み上げてやればよし．そこで高さを指定する必要が生じます．リスト3－4を見てください．初登場のrqはreadquoteの略で，⏎ が押されるまでにキーボード



```
?ハシラ
ヨコ?50
テイメンノ カクド"?30
タテヨ チヂメル ワリアイ?.7
タカサ?50
```

から入力されたものを返すプリミティブ．また，(　)を使っているのは，localというプリミティブの引数が基本的にひとつなもので，いくつもの変数をlocalにしたいときに用いる方法です（localの働きについては後述します)．

List 3-4

```
to ハシラ
(local "l "a "q "h)
splitscreen
type [ヨコ?] make "l rq
type [テイメンノ カクド?] make "a rq
type [タテヲ チヂメル ワリアイ?] make "q rq
type [タカサ?] make "h rq
cs
repeat :h / 2
 [テイメン :l :a :q
  pu fd 2 pd
 ]
end
```

(4) ディスクへの保存と読込み

せっかく作ったプロシージャも，電源を切ってしまえばそれでおしまい．フロッピーディスクドライブまで付いているSMC-777で，それではあまりに芸がありません．ディスクにプログラムをセーブしてこそ，始めてFDDの真価が発揮されます．そこで，ディスクにプログラムを格納するためのプリミティブが必要となるのですが，これは御存知のsave，書式としては，

save "ファイル名

となります．ここで注意したいのは，Logoがデータ(プロシージャも含め)を格納しておくメモリ中の場所をワークスペースと呼びますが，saveのときにはこのワークスペースの内容を全部まるごとディスクに格納してしまうことです．ですから不要のプロシージャはあらかじめ消しておく必要があり，そのためには，

er "プロシージャ名

とします．このerはeraseの略で，指定のプロシージャをワークスペースから消去する働きを持っています．

セーブするときに付けるファイル名は8文字以内の英数字で，このときは大文字・小文字の区別はされません．また，すでに存在するファイル名を指定した場合は，

File ×××× already exists

とメッセージが出され，セーブできんよ，と伝えてきます．それでは，さきほどの直方体のプロシージャをディスクにセーブしてみましょう．

? save "hashira ⏎

セーブが終了したら，本当に記憶されたかを確認してみることにしましょう．ワークスペースにはまだ内容が残されたままですので，まずこれを片付けてしまいます．

? erall ⏎

これでワークスペース内のすべてのデータは消え去ります．確認は，

? poall ⏎

ワークスペース内の全プロシージャの内容とグローバル変数が表示されるはずです(もちろん，ここでは何も出てきません)．しからば，ファイル名 hashira を読み込んでみることにしましょう．プリミティブは，これまた御存知のように load です．

load "ファイル名

この書式で全部OK．ではさっそく実行して，

?load "hashira ⏎
テイメン defined
ハシラ defined

ディスクドライブが作動して，メッセージが表示されることで，所定のプロシージャが読み込まれたことがわかります．プロンプト状態になったなら，もう一度 poall でワークスペース内を覗いてみてください．ちゃんと読み込まれたことが確認されるはずです．

このようなディスク回りのプリミティブとして，この他に，ディスクに保存されている全ファイル名を表示するための getfs があります．これはディスク内の Logo による全ファイル名を[　]でくくって並べて表示してくれます．また，ファイル名を変更したい場合は changef．書式は，

changef "新ファイル名 "旧ファイル名

となります．さらに増設ドライブを用いるときに，登録ドライブ（ドライブ名を省略したときに指定されるドライブ）を変更するための命令があり，

setd "ドライブ名：（例 setd "b:）

を用います．なお現在の登録ドライブ名を知るためには，defaultdを使います．

save	ワークスペースの内容をすべてディスクにセーブする
load	指定されたファイルをディスクからロードする
getfs	ディスク上のファイル名をリストにして返す
erf	ディスクから指定のファイルを消去する
changef	ファイル名を変更する
setd	登録ドライブの設定を行う
defaultd	登録ドライブ名を返す

表3-2　ディスクに関する命令のまとめ

po	指定されたプロシージャやグローバル変数の定義を表示する
pops	すべてのプロシージャの内容を表示する
pots	すべてのプロシージャのタイトルを表示する
pons	すべてのグローバル変数の内容を表示する
poall	すべてのプロシージャとグローバル変数の内容を表示する
erase(er)	指定されたプロシージャを削除する
ern	指定されたグローバル変数を削除する
erall	すべてのプロシージャとグローバル変数を削除する
nodes	未使用ノード数を返す
recycle	ワークスペース内のゴミ集めをする

表3-3　ワークスペースに関する命令のまとめ

(5) Logoのプログラム構造

Logoではすでに見てきたように，いくつものプロシージャが積み重なることによってプログラムが成立します．ひとつのプロシージャが段階的に他のプロシージャを呼び出し，いくつかに仕事を分担しながら全体の仕事をまとめあげるわけで，このようなスタイルを階層構造と呼んでいます．

いくつものプロシージャに分かれたプログラムのうち，他のプロシージャを呼び出すことはあっても，決して自分だけは呼び出されないというプロシージャが存在します．まァ，メインルーチンのようなものですが，Logoではこれを，トップレベルのプロシージャと呼んでいます．また，プロシージャの積み重ねが階層構造を作るとき，トップレベルから順を追って到達する仕事（プロシージャ）の到達順位をレベルと言い，順位を表す数字をレベルの深さと呼びます．Logoにおけるプログラミングは，こうした階層構造をいかに上手に作るかにおもしろさがあり，また逆に難しさにもなっています．これ以降，こうした階層構造を持ったいくつかのプログラムを登場さ

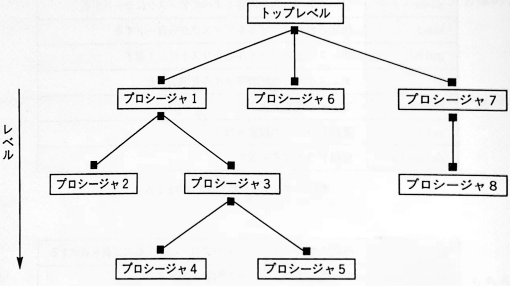


図3-3　プロシージャの階層構造

せ，Logoプログラミング上達の手懸りを作ってゆく予定です．しかし，その前にどうしても避けて通れぬ関門が，実は待ち構えています．それは数学嫌いの人間にとって，何とも恐ろしい引数や変数の話．でも，恐れることは決してありません．プログラムというひとつのストーリーの流れのなかにこうした数学的な要素を当てはめると，不思議なほどスムースに覚えることができるものです．考えてみれば，Logoはそんな思考の流れを作ることにより優秀な算数の先生にもなれるのです．まァ肩肘はらずにリラックスして，次の節に取り組んでみてください．

3-2　変数と引数

（1）引数・変数・定数

これまでにも，変数だの引数だのといったコトバはずい分と使われましたが，ここらでその実体を，少しばかり詳しく見てみることにしましょう．まずLogoの命令の書式を考えてください．これはただ一種類，

〈命令語〉〈引数〉〈引数〉………

これだけです．ですからたとえば，

print [SMC-777]

とした場合には，[SMC-777]はprint命令に対する引数になります．もうひとつ例を挙げると，

```
repeat 4 [fd 50 rt 90]
```

では，4と[fd 50 rt 90]がrepeatという命令の引数なのです．この引数にはここで見る4のように値そのものである定数の他に，自由に値を変えることのできる変数が使えます．この変数，さんざん試みたように，変数名の直前に：を付けることで識別されます．そして，変数に値を代入するときには，何度も使ったmakeに登場してもらうわけで，書式にすると，

```
make "〈変数名〉〈値〉
```

となるのですが，"（クォート）をつけないと意味をなしませんので，御注意のほどを．

（2）グローバル変数とローカル変数

Logoで使う変数には，グローバルとローカルという2種類があります．グローバル変数というのは，プロンプト状態からでもプロシージャのなかでも自由に代入したり参照したりできるもの．これに対してローカル変数とは，特定のプロシージャのなかのみで有効なもので，別のプロシージャのなかに同名の変数があっても関係しないものを指します．

具体的に考えてみましょう．プロンプト状態のとき，

```
?make "var  1 ⏎
```

とした場合，どのプロシージャのなかでも変数varは参照可能で，値を代入することもできます．しかし，プロシージャの冒頭（to～にあたる部分）にある変数は，そのプロシージャが呼び出されたときのみ引数が代入され，これはローカル変数となります．グローバル変数であれば，プロンプト状態のとき変数名を指定して⏎を押すことで変数の内容を見ることができますが，ローカル変数では不可能．プロンプト状能では見ることができません．

プロシージャのなかで何も宣言せずに変数を用いた場合，それはグローバル変数となります．逆に変数を使う前に，

```
local "変数名
```

を実行してやればローカル変数になるわけですが，これは宣言したプロシージャのなか及び，そのプロシージャから呼ばれたプロシージャのなかでも機能します．

```
to sample
local "var
make "var 13
sub
end
```

```
to sub
pr :var
end
```

この2つのプロシージャで試してみるとよくわかるのですが，sampleに呼ばれたsubのなかでもローカル変数varはそのままで，13という値が返されます．なおグローバル変数と同じ名前のローカル変数があった場合，ローカル変数が優先されますので注意してください．

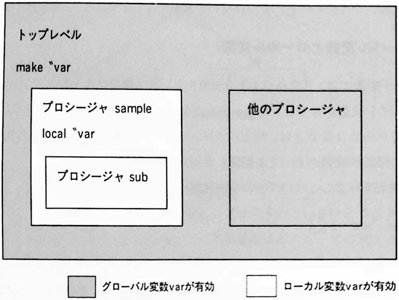


図3-4　ローカル変数とグローバル変数

(3) 値を返すプリミティブ

変数に付き物の値の話のついでに，値を返すプリミティブについても少し触れておきましょう．プリミティブには2種類 ── 何か機能を果たすものと値を返すものがあります．値を返すということは，こうしたプリミティブは引数に使えるということです．たとえば，引数の整数部分を返すintというプリミティブで考えると，

```
setalpha int 6.5
```

といった使い方ができるわけで，単に，

```
? int 7.5
7
```

という機能を見ただけではわからない応用法があるということです．

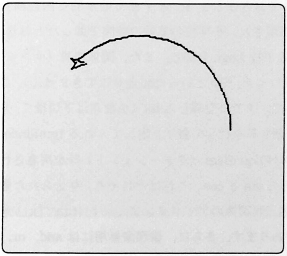
3-3 サンプル・アプリケーションズ

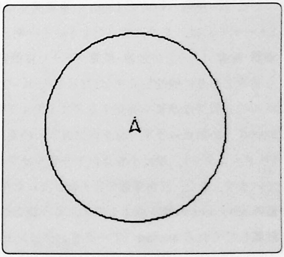
(1) 三角関数のワザ

タートル・グラフィックスの説明で，円を描くには多角形の考え方を応用すればよろしいと説明しました．しかし，実は円を描くにはこの他にも方法があるのです．昔なつかしい（なつかしかない，と言われる方もおられようが）三角関数，つまり sin と cos を用いた方法です．さっそくプロシージャをお見せしましょう．

List 3-5

```
to en1
;setposｦ ﾂｶｯﾀ ｴﾝ
ct cs
pr [ﾊﾝｹｲﾊ ｲｸﾂ (1 < 99) ?]
make "r rq
ct
make "x0 0
make "y0 0
make "a 0
pu rt 90 fd :r pd
label "zzz
  make "x (cos :a) * :r
  make "y (sin :a) * :r
  make "x1 :x0 + :x
  make "y1 :y0 + :y
  setpos se :x1 :y1
  make "a :a + 2
if :a > 361 [pu home pd stop] [go "zzz]
end
```





目新しい部分を説明してゆきましょう．まずプロシージャ名の次にある；(セミコロン)．プロシージャのなかにセミコロンがある場合，Logoはそれ以降の行末までを注釈のための文章と判断して何も実行しません．つまりセミコロン以下は自分のための備忘録のようなもので，トップレベルにこれを使ってプロシージャの内容をメモしておくと，後々なにかと重宝するものです．

次に出てくる見慣れないプリミティブはlabelです．これは，最後のあたりに見えるgoと一対をなすもの．Logoは，プロシージャの内部でgoを見つけると，同じ名前を持つlabelに無条件ですっ飛んでゆき，そこから実行を始めます．BASICのGOTO命令と同じような使い方と考えればよいわけです．つまり，同一プロシージャ内にgoとlabelの双方があって始めて機能するということです．ただ注意が必要なのは，goとlabelが同じ[　]のなかにあってはならないということです．また，seというプリミティブがありますが，これはsentenceの略で，ここではあとに続く：x1と：y1をまとめてひとつのリストにしています．

四則演算	関係演算	論理演算	三角関数	そ の 他
＋	＝	and	sin	int
−	＜	or	cos	round
＊	＞	not	arctan	remainder
／				quotient
				random
				rerandom

表3-4　演算プリミティブ一覧

ここで，説明ついでにLogoの演算機能に触れておきましょう．まず最も基本的な四則演算（＋−×÷）は，すでに用いたように＋−＊／で表現され，小学校の算数の時間で習ったとおり，乗算・除算（×÷）が加算・減算（＋−）に優先するのはLogoも同じ．また，関係演算（＝＞＜）も常識どおりに機能します(ただしBASICのように＜＝，＜＞といった組合せはできません)．これらの一般的な演算を補完するプリミティブとして，すでに登場したint（小数点以下切捨て）やround（小数点以下第一位を四捨五入）の他に，割り算を行って余りを出してくれるremainder（リメインダー），単に小数点以下を切り捨てるだけのquotient（クォーシェント）等が用意されています．また，三角関数で言えば，たった今使ったsinとcos，これはそれぞれ，与えられた数値のsin，cosの値を返すもので，もうひとつ，逆三角関数のアークタンジェント（$\tan^{-1}(x)$）を計算してくれるarctan（アークタンジェント）があります．さらに，論理演算用にはand，or，not（コンピュータの入門書に，イヤというほど出てきますなァ）があり，このほかに，あとで出

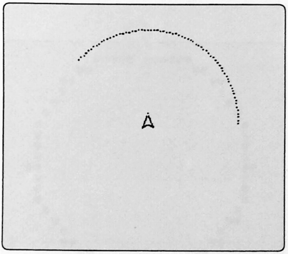
てきますが，乱数を扱うプリミティブの random（ランダム）と rerandom（リランダム）が，Logo における演算関係のプリミティブの代表格でしょう．

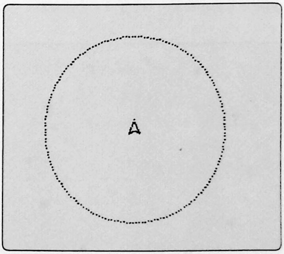
さて，en1 のプロシージャでは，setpos を用いるところがミソだったわけですが，今度は少し趣向を変えて，線以外のものを使って円を描いてみることにします．

List 3-6

```
to en2
;dotｦ ﾂｶｯﾀ ｴﾝ
ct cs
pr [ﾊﾝｹｲﾊ ｲｸﾂ (1 < 99) ?]
make 'r rq
ct
make 'x0 0
make 'y0 0
make 'a 0
label 'zzz
  make 'x (cos :a) * :r
  make 'y (sin :a) * :r
  make 'x1 :x0 + :x
  make 'y1 :y0 + :y
  dot se :x1 :y1
  make 'a :a + 2
if :a > 361 [stop] [go 'zzz]
end
```

ここで登場した新しいプリミティブは dot だけ．これは御想像どおり，指定された座標に点を打つ働きを持つもので，このときタートルの状態は変化しません．つまりタートルが遠隔操作で点を打ってゆくわけですが，描点の範囲はタートルの移動エリアと同じものです．



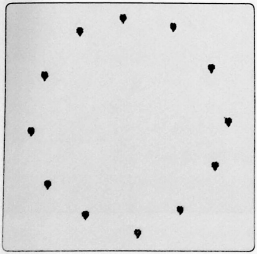


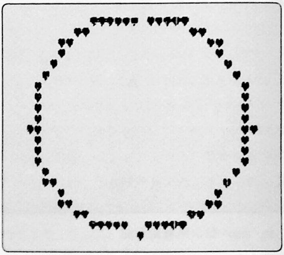
ドットで円を描けるなら，他の別なものを用いても円を描けるはず．何を使うかと申せば，文字コード表に出ている文字(キャラクタ)で，これはBASICもLogoも基本的に同じ．任意の文字をその文字コードから返させるにはchar（キャラクタ）というプリミティブがあり，charの次にコード表の数値を与えることで使えます．このcharを使い，テキストスクリーンに円を描いてみましょう．

List 3-7

```
to en3
:ﾃｷｽﾄｽｸﾘｰﾝ ﾉ ｴﾝ
textscreen ct
pr [ﾊﾝｹｲﾊ ｲｸﾂ (8 < 12) ?]
make 'r rq
ct
make 'x0 19
make 'y0 12
make 'a 0
label 'zzz
  make 'x (cos :a) * :r * 1.3
  make 'y (sin :a) * :r
  make 'x1 :x0 + :x
  make 'y1 :y0 - :y
  setcursor se :x1 :y1 type char 233
  make 'a :a + 30
if :a > 359 [stop] [go 'zzz]
end
```

このプロシージャで注意しなければならないのは，テキストスクリーン上に描画するということです．テキストスクリーンではX軸が縮んでいるため，補正をしなければなりません(1.2～1.3程度が適当．この場合は1.3で補正しました)．また，endの前に：a＋30と設定しましたが，この値が小さければもっと大量のキャラクターを画面上に表示できます．ただし余り小さくすると，円そのものがゆがんでしまうので注意してください．

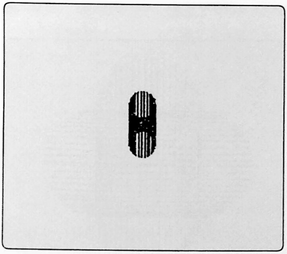


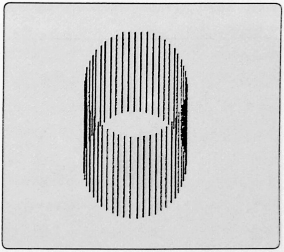


さて，これまでに覚えてきた(はずの)プリミティブと演算をもとにして，円を描くいくつかの方法を使ったプログラムを紹介しておきましょう．たいして難かしい手順を踏まなくとも，存外おもしろい図形を描けるという，Logoの特徴を如実に物語るものと思います．

List 3-8

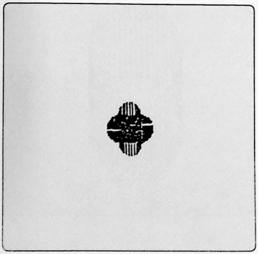
```
to en4
;ﾀﾃ ﾉ ｴﾝﾁｭｳ
ct cs
pr [ﾊﾝｹｲﾊ ｲｸﾂ (1 < 99) ?]
make "r rq
ct ht
make "x0 0
make "y0 0
make "a 0
setpc 6
label "zzz
  make "x (cos :a) * :r
  make "y (sin :a) * :r
  make "x1 :x0 + :x
  make "y1 :y0 + :y
  pu setpos se :x1 :y1 pd
  fd :r + 20
  make "a :a + 7
if :a > 361 [stop] [go "zzz]
end
```

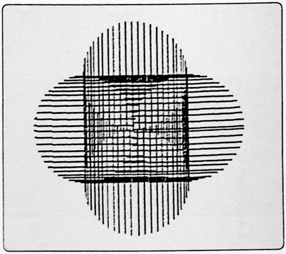




List 3-9

```
to en5
;ﾀﾃ ﾖｺ ﾉ ｴﾝﾁｭｳ
ct cs
pr [ﾊﾝｹｲﾊ ｲｸﾂ (1 < 99) ?]
make "r rq
ct
make "x0 0
make "y0 0
make "a 0
make "c 1
label "zzz
  make "x (cos :a) * :r
  make "y (sin :a) * :r
  make "x1 :x0 + :x
  make "y1 :y0 + :y
  pu
  setpos se (:x1 - :r) :y1
  pd
  setpc 1
  setx :x1 + :r
  pu
  setpos se :x1 (:y1 + :r)
  setpc 13
  pd
  sety :y1 - :r
  make "a :a + 7
if :a > 361 [stop] [go "zzz]
end
```





(2) 数当てゲーム

ここらでひと休み，頭の体操に簡単な数当てゲームでも試してみてください．プロシージャはいたって簡単，とにかく，まずは入力を．

List 3-10

```
to kazuate
ct
pr [ｶｽﾞｱﾃ ｹﾞｰﾑ]
pr [ｲｸﾂ ?]
make "m (random 100) + 1
make "x 1
label "cc
  make "n rq + 0 ........ rqにより入力されたワードを数値として比較するためには，このような演算を一度行う必要がある
  if :n > :m [print [ｵｵｷｲ !]]
  if :n < :m [print [ﾁｲｻｲ !]]
  if :n = :m
   [pr [ｱﾀﾘ !!!]
    (pr :x [ｶｲﾒﾃﾞ ｱﾀﾘ])
    wait 180
    kazuate]
  make "x :x + 1
go "cc
end
```

前の項でも触れた演算用のプリミティブが，このゲームの立役者を務めています．まず，乱数発生プリミティブのrandom．これは，引数として与えられた指定値に対し，0以上，指定値未満の範囲で乱数を発生する働きを持ちます(ただし指定値が32767を超えた場合はこの限りではなく，また指定値がマイナスの場合は同様に0以下の乱数になります)．randomは整数を乱数化するものですから，このゲームでは1～100までの数値を発生させるため，0～99までの整数を乱発生させ（random 100)，これに1を足し，1～100の整数値を得ています．次に出てくる演算記号は<>=の3つ．今さら説明するまでもないのですが，これらの記号を用いることで，オブジェクト同士の比較が可能となり，ここで使ったようにprintの引数を，ゲームの判定文として返すことができるのです．

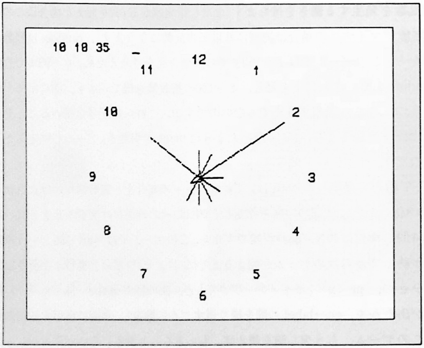
また，ここで始めて登場したwaitは，プロシージャの実行を一定時間だけ停止させるプリミティブ．このwaitのあとに任意の時間を指定してやれば，その時間だけ実行が止まっているわけです．この場合の時間の単位は約N／60秒程度ですから，このゲームではwait 180という指定なので，一度ゲームが終ってから次のゲームが始まるまでに，およそ3秒間ほど実行が中断されているわけです．もうひとつ，pr :x ［カイメデ アタリ］と，次の行のmake "x :x + 1は，何回かかって答えが出たかを，go→labelの間を繰り返すごとに勘定し，画面に表示する役割りを果たしています．このゲーム，もう少し頭を使えば，もっともっとおもしろいバリエーションができるもの．ひとつ，挑戦してみてください

```
?kazuate
カズ゛アテ ゲ゛ーム
イクツ ?
```

```
?kazuate
カズ゛アテ ゲ゛ーム
イクツ ?
50
チイサイ !
70
オオキイ !
62
チイサイ !
67
オオキイ !
66
アタリ !!!
5 カイメテ゛ アタリ
```

(3) Logo Clock

タートル・グラフィックスもかなりわかってきたし，演算も使えるようになり，そろそろDr. Logoのエキスパートに近くなってきました（と，書いておこう）．この辺で，いままでの集大成の意味を込め，少々複雑なプログラムに取り組んでみることにします．作ってみるのは，画面いっぱいを使った時計，それもアラーム機能の付いた目覚し時計で，デジタル表示まで付いてくる．時計屋を探したって，こんな珍しい時計はめったにお目にかからない．作るのにそれなりの芸当が必要なので，以下に留意点を書いておきます．



Logo CLock with alarm

まず，このLogoには時間関数が内蔵されていない，ということが問題として考えられます．そのために，この目覚し時計のプログラムでは，waitを使って，時間の単位を作ってみました．waitの単位数値で1／60秒プロシージャの実行が停止されることを利用して，時間を調整するわけです．なお，このプログラムでは最低単位は5秒きざみで，あまりアテにはなりませんが，分単位，時間単位は思いのほか正確で，実験してみたところ，6時間につき5分程度の誤差が生じるようでした．

次に注意したいのは，Dr. Logoではエディットバッファが思いのほか少ないため，ある程度の長さ以上のプログラムは，一度にエディット不可能ということです．そこで，このプログラムはリストを前半，後半の2つに分けて説明してゆきます．

それでは，さっそくプロシージャを紹介してゆきましょう．まずは前半について簡単に説明を加えます．

List 3-11

```
to time :nh :nm ;ｲﾏﾉ ｼﾞｶﾝ ﾌﾝ
cs setsplit 25 ct setborder 8
mt ct enen
label "start
  make "ns 0
  if :nm > 59
   [make "nh :nh + 1 make "nm 0]
  if :nh > 11 [make "nh 0]
  setpc 4
  rt ((30 * :nh) + int (0.5 * :nm))
  fd 50
  home
  setpc 2
  rt (6 * :nm)
  fd 80
  home
  setpc 3
  if (and :nh = :mh :nm = :mm)
   [music]
  repeat 12
   [setcursor [0 0]
     (type :nh :nm :ns [] [] [])
     make "ns :ns + 5
     fd 20 bk 20 rt 30
     wait 152
     recycle
   ]
  cs make "nm :nm + 1
go "start
end
```

さて，何のためのプロシージャか，おわかり願いましたか？　まず頭の方から見てゆくと，label "startと，次のmake "ns 0 以下の行，これがこのプロシージャの目玉商品です．何をするかと

申せば，この数行で1分ごとの変数の初期設定を行っているわけで，60分を超えたり12時を超えたりしたら，それぞれ0分，0時に戻さなくては時計にならない．そこでifを使ってnmとnhをそれぞれ59と11で大小比較しているわけです．ちなみに，5秒ごとに表示を変更しているのはrepeat 12以降の命令リスト．実際に時の過ぎるのをカウントしているのは，この命令リストのなかのwaitです(時刻がうまく合わないときは引数の152の値を調整してください)．この繰り返しを抜けると分を表す変数（nm）に1を加え，go "startで再び1分ごとの変数の初期設定に戻るわけです．

waitの下にrecycleというプリミティブが登場していますが，これはLogoがプログラム実行中にメモリ内に発生するいらなくなったデータを，強制的に整理する命令です．この命令を入れておかないと，Logoは必要に応じて適当なタイミングで，このデータの整理を行います(タートル・グラフィックス実行中に，ふとカメさんが休憩するのに気付いた方も多いでしょう)．これが適当な間隔で発生すると時刻に狂いが生じるので，強制的に実行しているのです．また，ここで初めて登場したプリミティブとして，最初の行にあるsetborderが挙げられます．これは読んで字のとおりで，ボーダーエリアの色を指定するプリミティブ，引数はsetpcと同じカラーコードに対応しています．では，続きを見てみましょう．

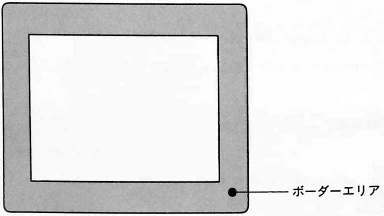


図3-5　ボーダーエリア

List 3-12

```
to mt
pr [ﾒｻﾞﾏｼ ｦ ｶｹﾏｽｶ ? (y / n)]
if equalp rc "y
 [type [ﾅﾝｼﾞ ﾆ ?] make "mh rq
  type [ﾅﾝﾌﾟﾝ ﾆ ?] make "mm rq]
 [make "mh 0 make "mm -1]
end
```

```
to enen
;ﾓｼﾞﾊﾞﾝ
setcursor [19 1] type "12
setcursor [12 2] type "11
setcursor [27 2] type "1
setcursor [7 6] type "10
setcursor [32 6] type "2
setcursor [5 12] type "9
setcursor [34 12] type "3
setcursor [7 17] type "8
setcursor [32 17] type "4
setcursor [12 21] type "7
setcursor [27 21] type "5
setcursor [20 23] type "6
end

to music
repeat 3 [tones [13 200 17 200 20 200]]
setalpha 14
end
```

これでおしまい，目覚し時計の完成です．見慣れないプリミティブは，まず rc. これは readchar (リードキャラクタ)の短縮形で，キーボードからの1文字入力を受け付けます．rq と似ていますが，rq がワードを返すのに対し，rc はキャラクタを返すもので，この rc が実行されると，キーボードからの入力があるまで，プロシージャの実行は中断されます．readchar は，readquote と違いリターンの入力を受けませんので，キーボードをひとつ押すとその押されたコードを返してきます．また，押されたコードに対応するキャラクターが画面に映る（これをエコーバックと呼びます）こともありませんので念のため．

もうひとつ新顔のプリミティブは，equalp(イコールピー)．与えられた2つのオブジェクトが等しいかどうかを判定します．等しければ，TRUE．そうでなければ，FALSE を返します．この equalp により，今説明したプリミティブ rc の値が y と等しいかどうか判定しているわけで，結果として関係演算子である＝と同じ働きをしているのです．

ところで，時計の文字盤をテキストスクリーンに描いていることに気付かれたでしょうか．これは，テキストスクリーンを使うことにより，cs の際にも文字が消えないようにしているのです．

なお最後のプロシージャに，music という名前を与えました．これは文字どおり音楽を担うプロシージャであることを意味し，目覚しの役割りを果たしています．ここで使われているtones(トーンズ）というプリミティブが音楽の演奏機能を持っており，そのあとに続く引数が，音程と音長を指定する音符のリストになっています．この音符で言うと，

```
tones [13  200  17  200  20  200]
       ↓   ↓    ↓   ↓    ↓   ↓
       ド  音長  ミ  音長  ソ  音長
```

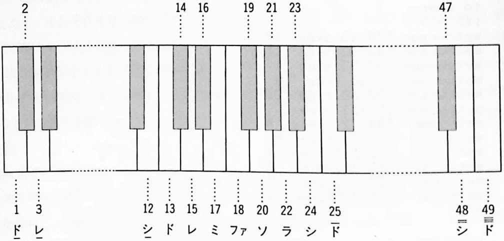


図3-6 tonesプリミティブにおける音階データ

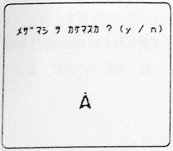
を3回繰り返すということで，各音程に加えて付けられた音長は，200／1000秒ずつであることを意味します（音長1単位は1／1000秒，msに相当します）．このmusicプロシージャが，トップレベルのtimeプロシージャのなかにある判定文，

```
if (and :nh = :mh :nm = :mm) [music]
```

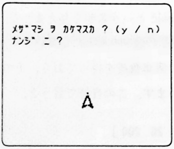
に対応しているわけで，andによって，目覚しをセットした時刻と現在の時刻が一致した場合に(TRUEのとき)，musicプロシージャを実行するのです．このmusicプロシージャには，setalpha 14が最後に付け加えられており，目覚しが鳴ったあとは時刻のデジタル表示も点滅させ，それと知らせるようにしてあります．それでは，実際に使ってみてください．

? time 現在時 現在分 ⏎

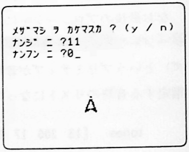
でスタートします．



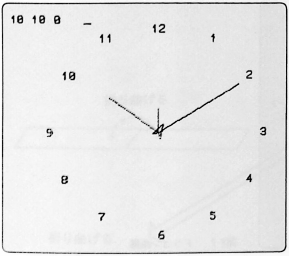
アラームを鳴らすかきいてくる



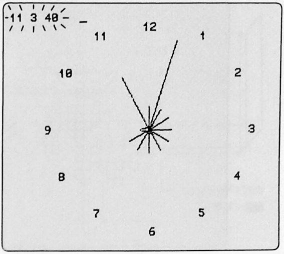
yを選んで



時刻を11時に設定



時計が働き始める



11時を過ぎるとデジタル表示が点滅する

(4) ドラゴン曲線を描く

これまでもたびたび試みたように，Logoではプロシージャのなかに，他のプロシージャを呼び出して，実行させることができます．つまり，ひとつのプロシージャをプリミティブのように使うことができるわけで，このようにプロシージャが他のプロシージャを呼び出して実行することを，プロシージャのコールと言います．また，すでに見てきたように，Logoにはgo～labelなどを使わなくとも，プロシージャのなかに自分自身を定義して，繰り返し実行を続けるという能力もあります．まァ，プログラムをひとつの輪に見立て，途中に適当に自分自身を置くことで，実行を繰り返してゆくと考えてよいでしょう．このような機能をリカーシブコール（再帰呼出し）と呼びます．このとき，何も指定しなければプロシージャは無限にプログラムを実行し続けますが（停止のためには CTRL + ESC を押す），終りの条件を正しく設定してやることで，有限回数の実行を得られます．また，自分自身をプロシージャ内で呼び出し，定義できるわけですから，使い方によっては，プロシージャに関数の機能を持たせることも可能となります．

List 3-13

```
to fact :n
if :n = 0 [op 1]
        [op :n * fact :n - 1]
end
```

リスト3-13に示した例は，op (output) という，プロシージャの実行を終了させて，得られた値を実行結果として返すプリミティブを用い，リカーシブコールを使うことで階乗を求めるプロシージャになっています．このような使い方をさらに発展して考えると，プロシージャ同士の呼び合いと再帰的呼び出しを組み合わせて使うこともできるようになります．

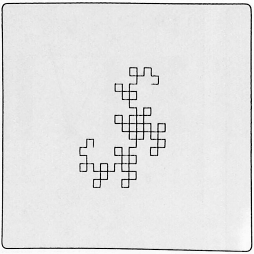


図3-7　ドラゴン曲線

次にドラゴン曲線を描いてみましょう．ドラゴン曲線は，ヒルベルト曲線，シェルピンスキー曲線と並び称されるメジャーな再帰曲線です．図3－8に示すように，n次のドラゴン曲線はn-1次のドラゴン曲線のまっすぐな部分を一端から順に右左右左と90度に曲げたものです．また，ドラゴン曲線を具体的に作る方法として，図3－9に示すように，細長い紙を何回か折り曲げ，折りしろを90度にするという方法があります．n回折り曲げて広げると，この紙の長い方の辺がn次のドラゴン曲線になります．

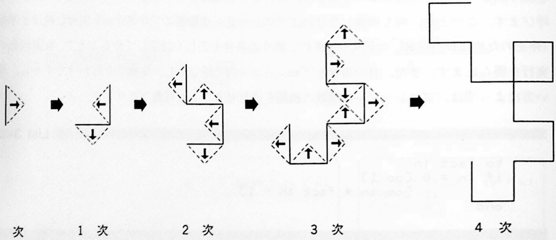


図3-8　0次から4次のドラゴン曲線

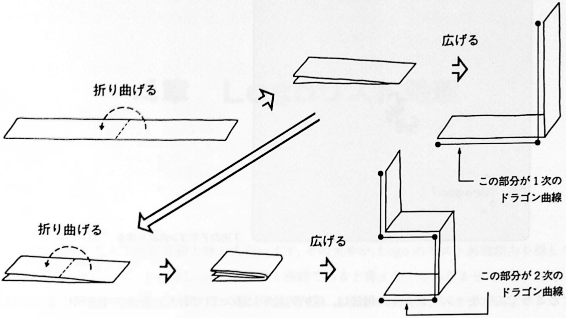


図3-9 紙を折って広げることでドラゴン曲線を得る

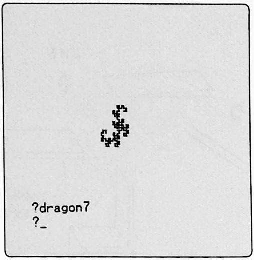
リスト3-14にドラゴン曲線を描くプログラムを示します．トップレベルは，dragonというプロシージャです．dragonrとdragonlが互いに呼び合っていますが，こういう再帰呼出しを，相互再帰といいます．図3-8をよく見て，プログラムを読んでみてください．

List 3-14

```
to dragon :order
cs ht
dragonr :order
end

to dragonr :order
if :order = 0
 [fd 2]
 [dragonr :order - 1
  lt 90
  dragonl :order - 1
 ]
end

to dragonl :order
if :order = 0
 [fd 2]
 [dragonr :order - 1
  rt 90
  dragonl :order - 1
 ]
end
```



7次のドラゴン曲線を得る

ところで，リカーシブコールの用法は，帰納的数学に少々似ており，普通のBASICなどには見られない機能です．LISP系の言語に特徴的なこの機能は，プロシージャの実行に際し，自分自身の定義を参照する定義の方法と考えることができます．ドラゴン曲線で見たように，コールとリカーシブコールを上手に組み合わせることにより，Logoにかなり大きな仕事をさせることが可能となります．この機能を使う使わないで，Logoによるプログラミングの限界は大きく変ってくるでしょうし，次章で述べるリスト処理にしたところで，リカーシブな用法がなければ威力が半減してしまいます．また，LISP系言語にとって，関数の再帰的定義は基本的な概念であるということも，覚えておいて損はないと思います．

Logoのプリンタ出力

Logoには，画面プリントアウト用プリミティブとして，copyonが用意されています．copyonが実行されると，それ以後，画面に出力されるすべての文字がプリンタに出力されます（editモードを除く）．通常，Logoのリストをプリントアウトする場合，copyonを実行したのちpo"プロシージャ名，あるいはpoallを使って，プログラムリストを画面に表示しながら，同時にプリンタにもこれを出力します．copyonを実行すると，以後画面に出力するものがそのままプリンタにも出力されます．この状態から抜け出すには，copyoffというプリミティブを使います．

このように，copyonは，画面に現れるすべての文字を印字する機能を持っているので，ときには不都合を起こすこともあります．この状態でプログラムリストを出力しようとすると，poallやcopyoffといったプログラムに関係ない文字が印字され，あまり体裁がよくありません．きれいなプログラムリストを印字しようとすると，Logoに頼らずにSony Filerにお願いしたほうが賢明です．Sony FilerのPRINTというコマンドを思い出してください．これは，アスキー形式で記録されたファイルの内容をプリントアウトする機能を持っています．システムメニュー画面上で指定のLogoファイルを選んでPRINTを実行すれば，余分なもののないきれいなプログラムリストが手に入ります．

4章　Logoリスト処理

Logoを称して人工知能言語と呼ぶ人がいます.その大半が,Logoのリスト処理能力を指してそう唱えているようで，これはLogoを象徴する機能であると言えなくもありません(もっとも，本格的にＡＩをやっている人が聞いたら腹を立てるでしょうなァ).実は，いままで試してきたプログラムの中にも，リスト処理を使ったものがいくつかありました．賢明な読者諸兄はすでにお気付きのことと思いますが，このリスト処理というシロモノ，無意識のうちに結構使えてしまうものなのです．それは，ひとつにはLogoの言語としての使い易さに起因するものと思われます(言ってみれば単純なのです）し，さらにはプログラミングの際，目的優先でプロシージャを組み立ててゆくという，Logoの特徴もまた，物を言っているのかもしれません.いずれにしろ，リスト処理を積極的に活用することで，情報の管理や検索といったデータ関係の処理，またちょっと気のきいた知的（に見える）なゲーム等に応用することができ，存外便利な機能であると思われます.

Logo関係の最後の章である本章では，このリスト処理に関する説明を行います．タートル・グラフィックスと並ぶLogoの特徴とも言えるリスト処理．しっかりマスターしてください．

4-1　リスト処理関連のプリミティブ

Logoの扱うデータの単位として，ワードとリストがあります.これらについては，すでに多少説明していますが,リスト処理の解説を始める前に，復習しておきましょう．

ワードとは要するに文字の集まりと考えればよくて，たとえば,

abcd

3456

3A4ア

といったものがワードです．記号のなかにはワードとして使えない（というよりは，使いにくいと言うべきか？)文字が14文字ほどありますが,これらについてはAPPENDIXに載せたエスケー

プ文字の説明を参照していただくとして，次にリストの説明．

さきほどからリスト処理という言葉を何度も使っていますが，このリストとは，ワードをひとまとめにしたものと定義できて，

[abc]
[Sony Filer]
[アカサタナ ABC 123]

などがリストです．リストには，中身が空ということもあって，これは，

[ ]

という形で表します．またリストの中にリストを置くことも可能で，

[abc def [ghi lmn]]

などという形のものもリストと呼びます．

実際にリスト処理を行う場合は，ワードが無意味な文字の並びになることはまずないわけで，リストとは，意味を持つデータが一定の目的に従って集積したものと考えられるでしょう．

リスト処理とは，乱暴に言ってしまうと，データの固まりのリストに対して，あれこれ手を出して操作を行うことを指すものです．Logoには，上述のようなリストを作り出したり，また任意のリストから目的に見合ったデータを取り出し，判定し，加工または削除したりという機能が備わっているため，任意のデータにダイレクトな加工を施すことができるわけです．まァ，具体的な説明をしないとわかりにくいので，とにかくリスト処理を試みるために，プリミティブを紹介してゆくことにしましょう．

(1) 要素関係のプリミティブ

データ，つまりワードなりリストなりを，オブジェクトと呼びます．このオブジェクトを構成する単位，リストであればワード，またワードであれば文字に当たりますが，これを要素と言います．つまりリストの要素はワード，ワードの要素は文字だと言い換えることができるでしょう（リストのなかにリストがあるときは，含まれる方のリストが要素になります）．オブジェクトから要素を入れたり出したりすることでリスト処理は初めて機能するわけで，まずはこの出し入れに関するプリミティブを，ひととおり紹介しましょう（重複もありますが，念のために全部を——）．

first

まず最初はfirstです．これはオブジェクトがワードであれば，最初のひと文字を返します．

```
?first "ascii ↵
a
```

これがリストになってくると，最初の要素を返すことになり，

```
?first [ascii no books] ↵
ascii
```

となりますし，リストのなかの先頭にリストがある場合ならば，でくわしたリストを返してよこします．

```
?first [ [books] wa ascii] ↵
[books]
```

さらにfirstを２つ続けて使用すると，最初にでくわした要素から，さらにその先頭の要素を取り出してきます．

```
?first first [ [ascii no books] wa honya de] ↵
ascii
```

butfirst

次にbf(butfirst)．見ただけで何となく働きがわかりそうなこのプリミティブは，最初に出合った要素を探し出し，それを除いたものを返します．

```
?bf [ascii no books] ↵
[no books]
?bf [ [ascii no books] wa honya de] ↵
[wa honya de]
```

item

３番目のitemは，指定番号を与えることで任意の要素を取り出してきますので，

```
?item 2 [ascii no books wa honya de] ↵
no
?item 2 [ [ascii no books] wa honya de] ↵
wa
```

これで要素関係のプリミティブは終り．リスト処理といっても，この程度ならたいしたことはありません．

では次に，ここまでのプリミティブを組み合わせたプロシージャを見てみましょう．次のプロシージャが何を意味するのか，おわかり願えますかな？

List 4-1

```
to elements :l
if emptyp :l
 [stop]
 [pr first :l
  elements bf :l
 ]
end
```

emptypは，要素数が0か否かを判断して返す働きを持つプリミティブ．つまり空のオブジェクトかどうかを判断するのです．このプロシージャを実行すると，

```
?elements "abc
a
b
c
?elements [ｱｶｻﾀﾅ ABC 123]
ｱｶｻﾀﾅ
ABC
123
?elements []
?elements [abc def [hij lmn]]
abc
def
hij lmn
?
```

リスト4-1の実行例

となります．つまり要素を順番に取り出し，出た順に並べてゆくものでした．では似たようなものをもうひとつ．

List 4-2

```
to last :l
if emptyp bf :l
 [op first :l]
 [op last bf :l]
end
```

ここで使っているopは，リカーシブコールの説明のときも出てきたように，プロシージャのなかだけで機能する命令で，その引数の値をプロシージャの結果として返し，プロシージャから抜け出すプリミティブです．そこでこのプロシージャは，オブジェクトから最後の要素を取り出して表示することになります．

```
?last "abc
c
?last [ｱｶｻﾀﾅ ABC 123]
123
?last [abc def [hij lmn]]
[hij lmn]
?
```

リスト4-2の実行例

(2) 加工関係のプリミティブ

次に，ワードやリストの合成，追加といった機能を持つプリミティブを紹介してゆくことにしましょう．

word

まずは，バラバラのワードをひとまとめにする（合成する）命令，wordから．

```
?word "SMC "777 ↵
SMC777
?(word "SEIKO "no "PASOKON) ↵
SEIKOnoPASOKON
```

wordは，基本的には2つのワードを引数とするプリミティブで，これを1つのワードにします．引数が2つ以外のときは，この例のようにプリミティブと引数全体を1組のカッコでくくる必要があります．

list

次にlist．このプリミティブも基本的に引数は2つで，この2つの値を要素とするリストを返します．引数が2つ以外のときは，wordと同様に全体をカッコでくくる必要があります．

```
?(list "SMC777)↵
[SMC777]
?list "matsuda "seiko↵
[matsuda seiko]
?(list "SEIKO "no "PASOKON)↵
[SEIKO no PASOKON]
```

この場合，引数のなかにリストが入っていてもかまいません．

```
?(list "SMC777 "wa [SEIKO no PASOKON] )
[SMC777 wa [SEIKO no PASOKON] ]
```

sentence

listと似たプリミティブにsentence(se)があります．listとの違いは，引数にリストがきたときにはっきりします．

```
?se "matsuda "seiko↵
[matsuda seiko]
?(se "SMC777 "wa [SEIKO no PASOKON] )↵
[SMC777 wa SEIKO no PASOKON]
```

つまり，引数にリストがきたときには，外側のブラケットをはずして，リストに合成するのです．次の例もよく見ておいてください．

```
?se [SMC777] [wa [SEIKO no PASOKON] ]↵
[SMC777 wa [SEIKO no PASOKON] ]
```

fput

加工関係の最後のプリミティブはfputです．これは，1つ目の引数を2つ目の引数の要素として，2つ目の引数の先頭に挿入するプリミティブです．seと似た働きと思われがちですが，fputの場合は要素を追加するためのプリミティブと考えておくと間違いないようです．

```
?fput "matsuda "seiko↵
matsudaseiko
?fput [SMC777] [wa [SEIKO no PASOKON] ]↵
[ [SMC777] wa [SEIKO no PASOKON] ]
```

(3) ワード処理関係のプリミティブ

ワードという，比較的小さなデータの単位を引数として，ちょっとした処理を行うプリミティブがあるので，紹介してゆきましょう．

asciiとchar

asciiというプリミティブは，与えられたワードの先頭の文字のキャラクタコードを引っぱり出してきます．キャラクタコードは，3章でも説明したとおり，文字に対応する数値のことで，BASICで使うキャラクタコードとほぼ同じです（APPENDIX参照）．

```
?ascii "a ↵
97
?ascii "ascii ↵
97
```

これに対してcharは，キャラクタコードを引数として，それに対応した文字を探してくるものです．

```
?char 97 ↵
a
?char 233 ↵
♥
```

lowercaseとuppercase

lowercase(lc) とuppercase(uc) は，共に以下に続くワード中のすべての文字を，それぞれ小文字ないし大文字に変換します．

```
?lc "ASCII ↵
ascii
?uc "ascii ↵
ASCII
```

これらのプリミティブは，すでに使ってきた，rq (readquote) やrc (readchar) と組み合わせて使うとひときわ威力を高めます．またデータの表示等の際にも重宝する機能です．

(4) 調査役プリミティブ

リスト処理を行う際は，ワードやリスト，あるいは数値（数字からなるワードをわざわざこう呼ぶ）などのデータの形式が，プリミティブもしくはプロシージャを実行させる上で重要な意味を持ちます．たとえば，引数が数値でなければならないプリミティブに数値でないものを与えても，データの形式が異なるために実行はされません(当り前だ)．このために，入力データや変数のチェックといったものが避けられないわけで，そのためのプリミティブが用意されているのです．

listp

listpは引数がリストか否かを判断するもので，

```
?listp "ascii↵
FALSE
?listp [ascii]↵
TRUE
```

のように，引数がリストのときはTRUE，そうでないときはFALSEという値を返します．

wordp

wordpはlistpと似たようなもので，判定対象がリストからワードに変るだけ．

```
?wordp "ascii↵
TRUE
?wordp [ascii]↵
FALSE
?wordp 1 ↵
TRUE
```

numberp

numberp はオブジェクトが数値として扱えるか否かを判定するもので，リストになっている場合は否と返してきます．

```
?numberp "007↵
TRUE
```

```
?numberp 007↵
TRUE
?numberp [007]↵
FALSE
```

emptyp

データ形式も重要ですが，リストやワードがどんな状態になっていて，要素がどんな具合なのか，というのも大切な要件となります．emptypは前述したとおり引数（オブジェクト）が空かどうかを判定します．

```
?emptyp [ ]↵
TRUE
?emptyp "↵
TRUE
```

equalp

2つの引数（オブジェクト）同士が相等であるか否かを判断するプリミティブはequalpです．これは＝という記号でも代用できて，

```
?equalp [ascii] [ascii]↵
TRUE
?= "ASCII "ASCII↵
TRUE
?equalp [ascii] "ascii↵
FALSE
?equalp [seiko] [SEIKO]↵
FALSE
```

となります．

memberp

1つ目の引数（オブジェクト）が2つ目の引数（オブジェクト）の要素かどうかを判定するプリミティブが，memberpです．この判定のあとにwhereを付け加えると，最初のオブジェクトが，次のオブジェクトの何番目の要素なのかまで調べることが可能です．

```
?memberp "se "seiko ↵
TRUE
?where ↵
1
?memberp [PASOKON] [SEIKO no PASOKON] ↵
FALSE
?memberp "PASOKON [SEIKO no PASOKON] ↵
TRUE
```

count

countは引数であるオブジェクトの要素の数を調べてくれるプリミティブです.

```
?count "matsuda ↵
7
?count [SMC777 wa [SEIKO no PASOKON] ] ↵
3
```

shuffle

最後に，調査役とは逆に，デタラメを作り出すプリミティブを紹介しておきましょう．与えられたリストの要素をデタラメ（ランダムと言えば聞こえはよいが）に並べ換え，もっともらしくリストの形で返す働きを持っています.

```
?shuffle [SMC777 wa SEIKO no PASOKON] ↵
[SEIKO SMC777 no wa PASOKON]
```

(5) データ管理プリミティブ

Logoには，生まれながらの情報管理者としての資質が備えられています．みずからのなかで情報を登録し，加工したり検索したりができるのです．それは主に，Logoがプロシージャや変数の内容を，プロパティ（性質といったような意味）として内部的に登録し，管理する能力を備えていることによります．プロパティは名前（登録名）とプロパティ名，それに内容（リスト，ワード，数値）によって構成され，読み書きは自由に行えます.

Logoにはユーザーが設定するプロパティのほか，システム自身で作るプロパティもあり，それがLogo自体を形成しているとも考えられます．つまり，変数を扱う変数プロパティや，プロシージャを扱う定義プロパティなどの，いわゆるシステムプロパティが，Logoを支えているわけです.

このシステムプロパティは合計10個あり，いずれも先頭にピリオドが付いた3つの大文字で表現されています（システムプロパティに使われているプロパティ名は，ユーザーが作るプロパティ名には使えません．詳しくはAPPENDIXを参照してください）．

プロパティを縦横に使うことで，Logoはそれ自身，データベースとしての機能を果たします．結構大きなデータを扱うことも可能なため，ぜひとも習熟しておきたいLogoの機能と言えるでしょう．では，プロパティ関係のプリミティブを，ざっと紹介してゆきます．

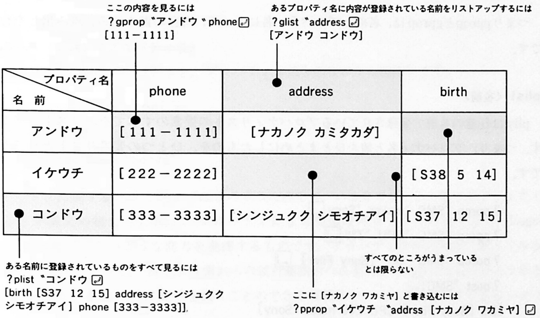


図4-1 プロパティの使用例

pprop 〈名前〉〈プロパティ名〉〈内容〉

pprop は，プロパティにデータを登録するためのプリミティブです．プロパティ名に内容を加えてリストにし，任意の名前のプロパティリストとして登録する機能を持っています．
たとえば，

```
? pprop "SMC "maker "Sony
? pprop "SMC "CPU "Z80
```

とすると，SMCという名前のプロパティリストは，

```
[CPU Z80 maker Sony]
```

となります．

gprop 〈名前〉〈プロパティ名〉

gpropは，プロパティのなかにあるデータを入手するためのプリミティブです．任意の名前に登録されているプロパティリストのなかから，指定されたプロパティの内容(リスト，ワード，数値)のみを取り出す機能を持ちます．

```
?pprop "SMC777 "maker "Sony⏎
?gprop "SMC777 "maker⏎
Sony
```

つまりppropとgpropは，名前とプロパティ名によって，内容の書き込み，読み出しを行うわけです．

plist 〈名前〉

plistは任意の名前で登録されているプロパティリストの要素のすべてを，リストの形で返します．つまり，プロパティ名と値をひとまとめにしたものを，ひとつの要素リストとして返すわけです．

```
?pprop "SMC "maker "Sony⏎
?pprop "SMC "CPU "Z80⏎
?pprop "SMC "OS [Sony Filer] ⏎
?plist "SMC⏎
[OS [Sony Filer] CPU Z80 maker Sony]
```

glist 〈プロパティ名〉

glistは任意のプロパティを持つ，名前のリストを返す働きを持ちます．最前の例に続けると，

```
?glist "maker⏎
[SMC]
?glist "CPU⏎
[SMC]
?glist "OS⏎
[SMC]
```

となるわけです．

pps

ppsは，ワークスペース内にある，ユーザーが定義した全プロパティ内容を，画面上に表示(のみ）する働きを持ちます．

```
?pps ⏎
SMC's OS is [Sony Filer]
SMC's CPU is Z80
SMC's maker is Sony
```

remprop 〈名前〉〈プロパティ名〉

remprop は任意のプロパティを，プロパティリストのなかから削除します．

```
?remprop "SMC "OS ⏎
?plist "SMC ⏎
[CPU Z80 maker Sony]
```

プロパティに関するプリミティブはこの6つだけです．いままで見てきた様々なプリミティブであるとか構文の組合せに加え，プロパティによる情報管理を上手に付加すると，Logo はBASIC には見られないような能力を発揮するものです．プロパティ管理は実際に使ってみると存外易しいものですので，習うより慣れろの試行錯誤のつみ重ねが大切なのではなかろうかと思います．とにかく数値以外に記号を扱うことのできる言語など，そう手軽に入手できないのですから．

次の節では，今まで見てきたリスト処理関係のプリミティブを利用したサンプルプログラムを紹介し，リスト処理のより具体的な使い方を身に付けていただこうと思います．

4-2 リスト処理サンプル・プログラム

では，いくつかのサンプルプログラムを見ていくことで，リスト処理，プロパティ利用などの方法を説明しましょう．まずは，簡単な電話帳プログラムです．

(1) 電話帳プログラム

まずは，次のページのリスト4-3を見てください．

電話帳に最低必要な項目は，名前と電話番号．この2つのデータをプロパティを使って管理しているのがこのプログラムです．トップレベルはtelというプロシージャで，メニューにより，登録，検索，すべての名前の表示，終了のいずれかを選びます．入力はrcで受けているので⏎を押す必要はなく，メニューに従って1文字入力してゆきます．

登録は，regというプロシージャ．ガイドに従って名前と電話番号を入れると，さきほど出てきたppropというプリミティブを使って，データがプロパティリストに登録されます．

```
pprop :name "tel :tel
```

List 4-3

```
to tel
ct
label "loop
  pr []
  setalpha 3
  pr [tel トウロク = push r key]
  pr [tel ケンサク = push s key]
  pr [ナマエノ リスト = push l key]
  type [シュウリョウスル = push q key]
  make "c lc rc
  pr []
  if memberp :c [r s l q] [pr :c]
  if :c = "r [reg]
  if :c = "s [src]
  if :c = "l [lst]
  if :c = "q [setalpha 7 stop]
go "loop
end

to reg
label "loop
  setalpha 6
  pr []
  type [name ?....] make "name rq
  if emptyp :name [stop]
  type [tel ?.....] make "tel rq
  if emptyp :tel [stop]
  pprop :name "tel :tel
go "loop
end

to src
label "loop
  pr []
  setalpha 3
  type [whose number?] make "name rq
  pr []
  if emptyp :name [stop]
  setalpha 6
  pr gprop :name "tel
go "loop
end

to lst
pr []
setalpha 6
pr glist "tel
end
```

:name が名前，次の "tel はプロパティ名，:tel は電話番号のデータとなります．

次に検索を行うのは，srcというプロシージャ．ここでは電話番号を知りたい人の名前を入れると，gpropにより，対応する電話番号を表示します．

```
gprop :name "tel
```

1つの名前には，1つの電話番号しか登録されていないので，名前に対応した電話番号が返されてきます．

なお，登録したすべての名前を表示するプロシージャがlstで，ここではglistを使っています．

```
glist "tel
```

これだけで，登録されているすべての人の名前を見ることができるわけです．

```
tel トウロク = push r key
tel ケンサク = push s key
ナマエノ リスト = push l key
シュウリョウスル = push q key
r

name ?....アンドウ
tel ?.....319-0539

name ?....イケウチ
tel ?.....336-9466

name ?....コンドウ
tel ?.....951-3368

name ?....
```

電話帳プログラムの実行例(登録)

```
tel トウロク = push r key
tel ケンサク = push s key
ナマエノ リスト = push l key
シュウリョウスル = push q key
s

whose number?イケウチ

336-9466

whose number?アンドウ

319-0539

whose number?
```

電話帳プログラムの実行例(検索)

```
tel ﾄｳﾛｸ = push r key
tel ｹﾝｻｸ = push s key
ﾅﾏｴﾉ ﾘｽﾄ = push l key
ｼｭｳﾘｮｳｽﾙ = push q key
l

ｺﾝﾄﾞｳ ｱﾝﾄﾞｳ ｲｹｳﾁ
```

電話帳プログラムの実行例（名前のリスト表示）

こんな簡単なプログラムですが，電話番号の変更は登録と同じ作業ででき，実際に利用することもできるわけです（データを入れたあとプログラムをsaveすれば，データも一緒に保存され，再びプログラムをロードしたときに再び参照，変更できます）.

（2）英文和訳プログラム

次に紹介するのは，極めてシンプルな英文和訳プログラムです.

List 4-4

```
to je
ct setalpha 7
pr [* * * * * * MY DICTIONARY * * * * * *]
pr []
pr [this, that, these, those]
pr [is, are]
pr [a, an, the, and]
pr [apple¥(¥-s¥), banana¥(¥-s¥), orange¥(¥-s¥)]
pr [strawberry¥(¥-ies¥), mine, yours]
pr [juicy, pretty, tasty, one, some, two]
pr [a_piece_of, my, your]
pr []
pr [* * * * * * * * * * * * * * * * * * * * *]
pr []
pr []
make "line rl
pr [] setalpha 6
make "n count :line
make "nm 1
label "loop
  make "w item :nm :line
  if :nm = 1 [syntax]
  make "wp glist :w
```

```
    if :wp = []
      [pr [] (pr :w [ハ,シリマセン。]) go "end]
    make "wp first :wp
    type gprop :wp :w
    make "nm :nm + 1
    if :nm = (:n + 1) [pr [デス。] go "end]
go "loop
label "end
setcursor [0 23] setalpha 11
pr [continue ? (y / n)]
if rc = "y [je]
setalpha 7
end

to syntax
if :n = 1 [stop]
make "head piece 1 2 :line
if memberp :head [[this is]
                  [that is]
                  [these are]
                  [those are]]
 [make "w word first :head item 2 :head]
end

pprop "f "this "コノ
pprop "f "that "ソノ
pprop "f "these "コレラノ
pprop "f "those "ソレラノ
pprop "e "juicy "ミズ"ケタップ"リナ
pprop "e "pretty "カワイラシイ
pprop "e "tasty "オイシイ
pprop "e "one "ヒトツノ
pprop "e "two "フタツノ
pprop "e "some "イクツカノ
pprop "e "a_piece_of "ヒトキレノ
pprop "e "my "ワタシノ
pprop "e "your "アナタノ
pprop "d "apple "リンコ"
pprop "d "apples "リンコ"
pprop "d "banana "ハ"ナナ
pprop "d "bananas "ハ"ナナ
pprop "d "orange "ミカン
pprop "d "oranges "ミカン
pprop "d "strawberry "イチコ"
pprop "d "strawberries "イチコ"
pprop "d "mine "ワタシノモノ
pprop "d "yours "アナタノモノ
pprop "c "a "
pprop "c "an "
pprop "c "the "
pprop "c "and "¥ ト¥
pprop "b "is "ハ,
pprop "b "are "ハ,
pprop "a "thisis "コレ
pprop "a "thatis "アレ
pprop "a "theseare "コレラ
pprop "a "thoseare "アレラ
```

これを人工知能的英文和訳プログラム……などと言うとおしかりを受けそうですが，プロパティを使った簡単な翻訳プログラムです．

Dr. Logoの乏しいスタックエリアのため，構文はbe動詞を使った現在形の英文を扱うことにして，まずLogoに英単語を教えましょう．ppropを使って，分類名，英単語，和文をプロパティリストに登録します．単語は，よく意味を考えて選ばないと翻訳した際，意味をなさなくなるので，もう一度初級英語のテキストを読み返すことを，お勧めします．

さて，翻訳を実行するメカニズムはいかに？ キーボードから入力された英文は，リストとして変数 :line に代入されます．これを，itemを使って頭から読み出し，この英単語 :w が属する名前 :wp を探します．

```
make "wp glist :w
```

この変数を，gprop文の引数に持ってくるのですが，gpropの引数は，ワード．glistで得られた :wp は，リストです．そこでfirst を使ってワードにしています．

これで，

```
gprop :wp :w
```

で，和文が検索されてくるわけです．そして，全英単語が変換され終ると，〝デス．〟が付いて文をしめくくるのです．これが，be動詞の現在形構文にした理由です．このプログラムには，構文解析機能がないので，助動詞の構文や過去完了形の構文などというのが出てくるとお手上げなのです．

ここでは，簡単な文法解析を試みました．this や that が，指示代名詞の場合と，主語代名詞として扱われる場合，つまり，thisが〝コレ〟のときと〝コノ〟のときを判断します．プロシージャsyntaxが，この機能を担っています．

```
* * * * * * MY DICTIONARY * * * * * *

this, that, these, those
is, are
a, an, the, and
apple(-s), banana(-s), orange(-s)
strawberry(-ies), mine, yours
juicy, pretty, tasty, one, some, two
a_piece_of, my, your

* * * * * * * * * * * * * * * * * * * *


this is my pretty apple

ｺﾚﾊ,ﾜﾀｼﾉｶﾜｲｲﾗｼｲﾘﾝｺﾞﾃﾞｽ｡

continue ? (y / n)
_
```

英文和訳プログラムの実行例

このように，プロパティリストと判定文によって文法解析をしたり，プロパティ名の付け方を工夫してさらには構文解析も可能です．

(3) 暗号とLOGOBASE

ここまでの2つのサンプルは理解できたでしょうか．まァ，プロパティに限らず，リスト処理のすべてに関して言えることに，とにかく慣れということが非常に重要だということを念頭に置いて欲しいと思います．BASICなどと違って，Logoのような LISP に近い言語では，やりつけないリストの処理という作業が必要で，これはもう慣れる以外にテはないのです．

以下に2つのプログラムを掲載します．前者のANGOは，漢字で書けば暗号で，読んで字の如く普通のコトバを暗号にするプログラム．ミソはshuffle文を使ってリストのなかのワードを乱数でデタラメに置き換えて表示し，いかにも暗号らしく見せるところ（もっとも，ちょっと頭をひねれば誰でも解けるような暗号ではあるが……)．また，あとに掲載するLOGOBASEの方は，これも読んだとおりのデータベース・プログラムで，さきほどの電話帳プログラムを少々モディファイし，名前と電話番号だけではなく，住所と誕生日もデータとして登録できるよう付加したもの．そのうえ削除や保存の機能も付けてあります．一応はデータベースらしく見えるものと思いますが，機能として最も重要な役割りを担っているのは，やはりプロパティ管理です．

リストと実行例を示しますので，是非プログラムを読み，777オーナーの方は打ち込んで，実行してみてください（説明をしていないプリミティブも少し出てきますが，リストのコメントとAPPENDIXのプリミティブ一覧を参照してください)．

List 4-5①

```
to ango
ct
setalpha 6
pr [* * * ｱﾝｺﾞｳ ｻｸｾｲ･ｶｲﾄﾞｸ ﾌﾟﾛｸﾞﾗﾑ * * *]
mktable                                         } 暗号表をつくるルーチンを呼ぶ
label 'menu
  setalpha 6
  pr []
  pr [1 = ｱﾝｺﾞｳ ｻｸｾｲ]
  pr [2 = ｱﾝｺﾞｳ ｶｲﾄﾞｸ]                          } メニューの表示と選択
  pr []
  setalpha 3
  type [ﾊﾞﾝｺﾞｳﾃﾞ ｴﾗﾝﾃﾞ ｸﾀﾞｻｲ¥ ]
 label 'inp
  make 'i rc
  if :i = '1
   [pr 1
    pr []
    pr [ﾓﾄﾆﾅﾙ ﾌﾞﾝｦ ﾆｭｳﾘｮｸｼﾃ ｸﾀﾞｻｲ]                } 暗号作成の処理
    setalpha 2
    make 'plain rq
    setalpha 7
    pr encrypt :plain
   ]
```

```
    [if :i = "2
      [pr 2
       pr []
       pr [ｱﾝｺﾞｳﾌﾞﾝｦ ﾆｭｳﾘｮｸｼﾃ ｸﾀﾞｻｲ]
       setalpha 4
       make "crypt rq
       setalpha 7
       pr decrypt :crypt
      ]
      [go "inp]
    ]
   setalpha 3
   pr []
   type [ﾂﾂﾞｹﾏｽｶ? (y / n)]
   if rc = "y [pr [] go "menu]
 setalpha 7
 end
```

（[if :i = "2 …] の部分）暗号解読の処理

List 4-5②

```
to mktable
(local "i "dummy)
setalpha 3
pr []
type [ｷｰ ﾄ ﾅﾙ ｶｽﾞｦ ﾆｭｳﾘｮｸｼﾃ ｸﾀﾞｻｲ¥ ]
rerandom                                  ) 乱数を初期化する
repeat rq [make "dummy random 1]          ) 入力された回数だけ乱数を発生させ更新する
setalpha 11
type [ｼﾊﾞﾗｸ ｵﾏﾁｸﾀﾞｻｲ]
make "i 32
make "table []
repeat 127 - 32
 [make "table se :table char :i
  make "i :i + 1
 ]
make "table shuffle :table                ) 文字のリストをシャッフルする
setalpha 3
setcursor [0 3]
pr [ｼﾊﾞﾗｸ ｵﾏﾁｸﾀﾞｻｲ]
end

to encrypt :plain                         ) 引数として原文:plainを受け取る
(local "crypt "i)
make "i 1
make "crypt "
repeat count :plain
 [make "crypt
    word :crypt
      item (ascii item :i :plain) - 31
           :table
  make "i :i + 1
 ]
op :crypt                                 ) 暗号文を返す
end
```

（make "i 32 〜] の部分）キャラクタコード32〜126までの文字のリストを作る

（[make "crypt 〜] の部分）原文の:i文字目を暗号に変換してそれまでに変換された暗号文:cryptにつなげる

```
to decrypt :crypt                          ) 引数として暗号文:crypt を受けとる
(local "i "plain "p)
make "i 1
make "plain "
repeat count :crypt
 [if memberp item :i :crypt :table
   [make "p where
    make "plain word :plain
    char :p + 31                           暗号文の:i文字目を解読し，それまでに解読さ
   ]                                       れた文:plainにつなげる
  make "i :i + 1
 ]
op :plain                                  ) 解読文を返す
end
```

```
* * * アンコ゛ウ サクセイ・カイト゛ク フ゜ロク゛ラム * * *

キー ト ナル カス゛ヲ ニュウリョクシテ クタ゛サイ 7
シハ゛ラク オマチクタ゛サイ

1 = アンコ゛ウ サクセイ
2 = アンコ゛ウ カイト゛ク

ハ゛ンコ゛ウテ゛ エランテ゛ クタ゛サイ 1

モトニナル フ゛ンヲ ニュウリョクシテ クタ゛サイ
I am Dr. Logo.
li-Vi(D=iu´f´=

ツツ゛ケマスカ? (y / n)_
```

暗号プログラムの実行例
(暗号作成)

```
* * * アンコ゛ウ サクセイ・カイト゛ク フ゜ロク゛ラム * * *

キー ト ナル カス゛ヲ ニュウリョクシテ クタ゛サイ 7
シハ゛ラク オマチクタ゛サイ

1 = アンコ゛ウ サクセイ
2 = アンコ゛ウ カイト゛ク

ハ゛ンコ゛ウテ゛ エランテ゛ クタ゛サイ 2

アンコ゛ウフ゛ンヲ ニュウリョクシテ クタ゛サイ
li-Vi(D=iu´f´=
I am Dr. Logo.

ツツ゛ケマスカ? (y / n)_
```

暗号プログラムの実行例
(暗号解読)

List 4-6①

```
to logobase
(local 'comd 'exe)
;
label 'start
cleartext
setcursor [5 4]
print [* * * * * LOGO BASE * * * * *]
setcursor [5 8]
print [ﾅﾆ ｦ ｼﾀｲﾃﾞｽｶ? No.ｷｰ ｦ ｵｼﾃｸﾀﾞｻｲ!]
setalpha 7
setcursor [0 10]
pr [¥ ¥ ¥ ¥ ¥ ¥ 1 = ﾄ ｳ ﾛ ｸ]
pr []
pr [¥ ¥ ¥ ¥ ¥ ¥ 2 = ｹ ﾝ ｻ ｸ]
pr []
pr [¥ ¥ ¥ ¥ ¥ ¥ 3 = ﾃﾞｰﾀ ｼｮｳｷｮ]
pr []
pr [¥ ¥ ¥ ¥ ¥ ¥ 4 = ｵ ﾜ ﾘ]
;
label 'loop
  setcursor [34 21]
  print [¥[ ¥]]
  setcursor [35 21]
  make 'comd readchar
  type :comd
    if :comd = '4
      [
        print []
        type [ﾃﾞｰﾀ ｦ ｾｰﾌﾞｼﾏｽｶ (y / n) ¥ ]
        make 'comd readchar
          if not :comd = 'n
            [
              type :comd
              erf 'logobase
              save 'logobase
            ]
        delete_pps
        cleartext
        setcursor [15 12]
        print [ﾏ ﾀ ﾈ ! ! !]
        setcursor [0 24]
        run [erall stop]
      ]
   if not memberp :comd [1 2 3]
     [go 'loop]
     [
       make 'exe (item where [makedata search erdata])
       load :exe
       run (list :exe)
       erase :exe
       go 'start
     ]
end
```

メニュー画面の表示（label 'start ～ pr [¥ ¥ ¥ ¥ ¥ ¥ 4 = ｵ ﾜ ﾘ]）

メニュー番号の入力（setcursor [34 21] ～ type :comd）

終了処理
データをセーブする場合，まず古いファイルを消去し，登録されたプロパティと共に自分自身をセーブしなおす
delete_ppsを呼んでプロパティを消去し，ワークエリアを初期状態に戻して終了する

メニューにないキーが押されていたとき，入力に戻る

メニューで選ばれたプログラムをロードし，実行後不要となるので消去する
ここではメモリを節約するために，makedata, search, erdataの3つのプロシージャは，それぞれ同名のファイルに別々にセーブしておく

```
to print_data          ……… "phoneの登録されているプロパティの名前を
clear text                  すべて表示するプロシージャ
setcursor [4 2]             "makedata、"search、"erdataの各プロージ
print [* * * * * ﾄｳﾛｸｻﾚﾃｲﾙ ﾋﾄ * * * * *]     ャから呼ばれる
setcursor [2 5]
type glist "phone
end

to delete_pps                      } すべてのプロパティを消去するプロシージャ
(local "name "i)
make "name glist "phone            } プロパティの名前のリストを得る
make "i 1
repeat count :name [
   remprop item :i :name "phone    |
   remprop item :i :name "address  | :i番目のプロパティをすべて消去する
   remprop item :i :name "birth    |
   make "i :i + 1
  ]
stop
end
```

List 4-6②

```
to makedata                        } プロパティを登録するプロージャ
(local "name "phone "address "birth)
;
label "loop
  print_data
  print []
  print []
  print [ﾅﾏｴ ｦ ｲﾚﾃｸﾀﾞｻｲ]
  make "name readquote
  if :name = " [stop]              } 何も入力しないとこのプロージャを終了する
  ;
  print [ﾃﾞﾝﾜ ﾊﾞﾝｺﾞｳ ｦ ｲﾚﾃｸﾀﾞｻｲ]
  make "phone readlist
  ;
  print [ｼﾞｭｳｼｮ ｦ ｲﾚﾃｸﾀﾞｻｲ]
  make "address readlist
  ;
  print [ｵﾀﾝｼﾞｮｳﾋﾞ ｦ ｲﾚﾃｸﾀﾞｻｲ]
  make "birth readlist
  ;
  pprop :name "phone :phone        |
  pprop :name "address :address    | プロパティを登録する
  pprop :name "birth :birth        |
go "loop

end
```

List 4-6③

```
to search                                    } 検索をするプロージャ
(local 'who 'all)
;
label 'loop
  print_data
  print []
  print []
  type [¥ ﾀﾞﾚﾉ ｺﾄ ｦ ｼﾘﾀｲﾃﾞｽｶ ?]               } 検索したい名前を入力する
  make 'who readquote
  if :who = ' [stop]                         } 何も入力しないとこのプロージャを終了する
  make 'all plist :who                       } 入力されたプロパティリストを得る
  if :all = []
    [
      print []
      type :who
      print [ｻﾝﾊ ﾄｳﾛｸｻﾚﾃｲﾏｾﾝ]                 } 入力された名前のプロパティがない場合
      print []
      type [ﾅﾆｶ ｷｰｦ ｵｼﾃｸﾀﾞｻｲ]
      make 'who readchar
      go 'loop
    ]
  print []
  print []
  (print :who 'ｻﾝﾉ)
   type 'ﾃﾞﾝﾜﾊﾞﾝｺﾞｳﾊ¥
  (print gprop :who 'phone)
   print []                                  } 指定された名前のプロパティの内容を出力する
   type 'ｼﾞｭｳｼｮﾊ¥ ¥ ¥ ¥ ¥
  (print gprop :who 'address)
   print []
   type 'ﾀﾝｼﾞｮｳﾋﾞﾊ¥ ¥ ¥
  (print gprop :who 'birth)
   print []
   type [ﾅﾆｶ ｷｰｦ ｵｼﾃｸﾀﾞｻｲ]
   make 'who readchar                        } 1回キーを押すと"loopに戻る
go 'loop
end
```

List 4-6④

```
to erdata                                    } プロパティを消去するプロージャ
(local 'name 'select 'i)
;
label 'start
print_data
print []
print []
type [ｾﾞﾝﾌﾞｹｼﾏｽｶ? (y / n) ¥ ]
make 'select readchar
pr :select
if not memberp :select [y n]                 } 入力がyでもnでもない場合，最初に戻る
  [go 'start]
if :select = 'y
  [delete_pps stop]                          } すべてのプロパティを消去するプロージャを呼ぶ
```

```
    [
     print []
     type [ダ゛レ ノ デ゛ータ ヲ ケシマスカ?]
     make "name readquote
     remprop :name "phone
     remprop :name "address
     remprop :name "birth
     stop
    ]
  end
```

入力された名前のプロパティを消去する

```
  ＊ ＊ ＊ ＊ ＊ LOGO BASE ＊ ＊ ＊ ＊ ＊

 ナニ ヲ シタイデ゛スカ? No.キー ヲ オシテクダ゛サイ!

  1 = ト ウ ロ ク

  2 = ケ ン サ ク

  3 = デ゛ータ ショウキョ

  4 = オ ワ リ
```

LOGOBASE(メニュー)

```
    ＊ ＊ ＊ ＊ ＊ トウロクサレテイル ヒト ＊ ＊ ＊ ＊ ＊

  アンド゛ウ イケウチ

ナマエ ヲ イレテクダ゛サイ
コンド゛ウ
デ゛ンワ ハ゛ンコ゛ウ ヲ イレテクダ゛サイ
951-3368
シ゛ュウショ ヲ イレテクダ゛サイ
シンシ゛ュクク シモオチアイ
オタンシ゛ョウビ゛ ヲ イレテクダ゛サイ
S37 12 15_
```

LOGOBASE(登録)

```
    ＊ ＊ ＊ ＊ ＊ トウロクサレテイル ヒト ＊ ＊ ＊ ＊ ＊

  コンド゛ウ アンド゛ウ イケウチ

 ダ゛レノ コト ヲ シリタイデ゛スカ ?コンド゛ウ

コンド゛ウ サンノ
デ゛ンワハ゛ンコ゛ウハ 951 - 3368

シ゛ュウショハ      シンシ゛ュクク シモオチアイ

タンシ゛ョウビ゛ハ    S37 12 15

ナニカ キーヲ オシテクダ゛サイ_
```

LOGOBASE(検索)

（4）まとめ

Logoの解説は，以上をもって終了します．いかがでしょう．Logoのおもしろさ，わかっていただけたでしょうか．

ユニークな教育用言語としてのLogoの将来は，ひとえに社会一般が，Logoの目指す学習過程創造型の教育方法を，認識し評価するか否かにかかっていると思われます．Logoによる教育では，コンピュータは学習環境を提供することに主眼が置かれているのです．そしてLogoの持つ理念とそれによって得られるメリットは，子供をひとりの知性ある人間として見た場合，少なくとも現代のマス教育の画一人間を作り出すようなシステムよりは有益なものと言えるでしょう．

このLogoが目指す教育方法が広がるには，まずLogoが社会的に認知されることが重要なファクターとなるのではないかと考えられます．そのためには，ひとりでも多くの人がLogoに触れ，そこからLogoのもたらすメリットを多重的に引き出してゆく必要があります．そして，それにはなにより，Logoをよりよく理解し，使いこなすことのできる人間が多数輩出せねばなりません．ですから，Logoが社会的に何らかの意味を持つ存在になり得るか否か，本章をお読みのあなたに負う部分が多分にあるのです．もっとも，難しく考える必要はなく，要はLogoを使い，プログラミングを通してLogoを知っていただければよし，結論はおのずと下されるものですから——．

Logoのもうひとつの側面，リスト処理に代表される人工知能的な言語としての特性にも，若干触れておきましょう．ファイゲンバウムの『第五世代コンピュータ』でずい分有名になった日本のＡＩ開発は，しかしよくよく見れば，まだまだ端緒のそのまたとば口に着いたばかり．本格的なＡＩの始動は，実はこれから始まるのだと言ってもよいでしょう．

ＡＩが実用に供された場合の社会的インパクト——それはパソコンの普及の比ではないと考えられます——を予測するなら，ＡＩ自体がより手軽な存在に近付かねばならないのは自明の理．コンピュータの発達が，ハードの面では草創期の先達の努力の賜物（たまもの）であったのに対し，ソフト面は幾多の名も知れぬプログラマの創意の累積で成り立ったことを鑑（かんが）みるまでもなく，ＡＩが一部の専有物であってはならないのは当然のことと言えるでしょう．

私たちがＡＩに近付くひとつの手段がプログラミング，それもLogoで試みたような，記号処理を実践するものです．すでにＡＩ言語として著名なLISPやPROLOG等の言語がパソコンレベルで使えるまでになってはいますが，問題はいずれも比較的高額で，しかも専門家やマニア以外には近付きにくい雰囲気を持っていること．その点で，Logoはリスト処理の入門に，最適の環境を提供してくれる存在ではないでしょうか．この章で試みたように，Logoにおけるリスト処理は，半ば無意識のうちに実行されてしまうほど自然なもの．Logoを用いたプログラミングは，近い将来のＡＩの登場に対処するための，自己啓発的学習になり得るのかもしれません．

まァ，小難しいことを考えるよりも，やはりLogoは，プログラムを楽しむことが一番でしょう．やれエリアをくい過ぎるだの，何をやらせてもトロいだの，Logoに対するケチはいくらでも付けられます．しかし，それでもなお，使って楽しい言語であるのがLogoのようです．そして，そんななかで，何か得るものがあるのなら，それにこしたことはないでしょう．

5章　777 BASICへの招待

システムメニューからBASICを選ぶと，ソニー自社開発による777 BASICが起動されます．このBASICは他機種に見られぬ豊富な機能が盛り込まれ，かなり強力かつ使い勝手のよいものに仕上がっています．しかし，多彩な機能が付加されたということは，それだけ覚えなければならないコマンドだの約束事が増えたということ．

本章では，まず777 BASICの特徴を述べ，続いて効果音やミュージックの鳴らせ方，ディスクファイルの扱い方といった，777 BASICを有効に使うための命令の説明，さらにMEMOで作ったデータのBASICでの利用法と説明を進めます．

777 BASICの文法の多くは現在最も普及しているマイクロソフトBASICに準じています．そこで本書でのBASICの解説は，マイクロソフト系のBASICの基本的な知識は一応持っているという方を想定して書いてありますが，多少なりとも難しそうな事柄は基礎から説明してあります．初心者の方も巻末に記した参考文献やマニュアル(SMC-777本体に付いてくるBASICハンドブック）を参考に本章を読みこなしてください．そうすれば，ハンドブックの巻頭に書いてあるように〝いつのまにかコンピュータと気持ちが通じ合う〟ようにもなっていくことでしょう．

5-1　777 BASICの特徴

777 BASICは，ソニーが初めて開発したパソコンSMC-70のディスクBASICの特徴を受け継いだ，ソニー独自のBASICです．本節では，まずこの777 BASICの特徴を紹介することにしましょう．

(1) 操作性

SMC-777は，操作性に多くの配慮がなされたパソコンで，その点を指してソニーみずからソフトウェア･プレーヤーと呼ぶほどです．777 BASICにおいても，操作性への配慮は十分に行われています．

プログラミング中リターンキーで文字色が変る

BASIC の起動後に入力を始めてみて，まず最初に気付くのは，キー入力のエコーバックが水色の文字で表示されているということでしょう．この水色表示は，RETURN キー(以後 ↵ で示す)で白に変ります．つまり命令がメモリに記憶されたことを視覚的に教えてくれるわけで，なかなか気のきいたアイデアだと言えるでしょう．たとえばスクリーンエディタでプログラムを修正するようなとき，リスト表示が白い文字であるのに対し，修正目的で新たに入力した文字がストアされるまで水色で表示されているため，修正箇所が非常にわかり易く，またタイプミスや入力のし忘れ等のチェックにも役に立ち，なによりも ↵ の押し忘れ(これがよくあって，いつも泣かされていたりして……) 防止に威力を発揮するものです．

キーファンクション

キー	機能
CLR	カーソルのある行のカーソル以降を消す
SHIFT + CLR	カーソルの行のカーソルより前を消す．残った文字は行頭までつめられる
CTRL + CLR	テキスト画面全体を消し，カーソルを左上端に移す
DEL	カーソルの位置の文字を消し，それより後をつめる
CTRL + DEL	カーソルのある行をなくして，それより下の行を上げる
INS	カーソルの位置にスペースを挿入する
SHIFT + INS	挿入モードの設定・解除．解除は ↵ を押すことでもなされる
CTRL + INS	カーソルのある行以降をひとつ下げ，空白だけの行を挿入する
HOME	行の頭頭にカーソルを移す
SHIFT + HOME	行の最後にカーソルを移す
CTRL + HOME	画面左上の端にカーソルを移す
SHIFT + ←	HOME と同じ
SHIFT + →	SHIFT + HOME と同じ
BS　RUBOUT	カーソルのすぐ左の文字を消しカーソル以降をつめる
CTRL + P	これを押すと画面に出たものがプリンタにも出力されるようになる．これはトグルになっていて，もう一度押すと解除される

CTRL + H	BS と同じ	CTRL + N	CTRL + CLR と同じ
CTRL + I	TAB と同じ	CTRL + O	CTRL + INS と同じ
CTRL + J	LF と同じ	CTRL + Q	CTRL + DEL と同じ
CTRL + M	↵ と同じ	CTRL + T	CTRL + HOME と同じ

表5-1　主なキーファンクション

777 BASICでは，[SHFT]や[CTRL]のキーと，[CLR]，[DEL]，[INS]，[HOME]それにカーソルパドル（上下左右それぞれ[↑]，[↓]，[←]，[→]で表す）などのキーの組合せで，コマンドやリスト入力時にかなりの働きをさせることが可能です．

まァ，豊富な機能が用意されているということは，それだけに慣れが要るということですので，表5-1に示した主なキーファンクションを頭に入れ，オペレーションのいかなるときでも使えるように訓練してください（有効に使うと，こんな力強い味方もないですゾ）．ただ注意しなければならないのは，プリントアウト時に用いる[CTRL]＋[P]をプリンタをつながずに押して，そのあと何かを入力して[↵]したりすると，それ以降キー入力を一切受け付けなくなってしまうこと．こんな場合はもう一度[CTRL]＋[P]を押して，プリンタ出力状態の解除を計らなくてはなりません．

なお，ファンクションキーも5つ用意されていて，他機種同様，キータッチの回数減少に役立ちます．BASIC起動時にファンクションキーに登録されている命令は表5-2のとおり．もちろん登録内容を変更する命令（DEFKEY）も存在します．

ファンクションキー	ノーマル	Shift
[F1]	List	Wipe' [↵]
[F2]	Edit	Console
[F3]	Auto	Gclear
[F4]	Debug	Renum
[F5]	Run	Keylist' [↵]

表5-2　ファンクションキーに登録されている命令

リストの行の分割と結合

777 BASICでは，プログラムの1行を2つに分けたり，2行を1つに結合したりすることも簡単にできますので，その方法を紹介しましょう．

まず，分割について．画面が次のようになっていたとします．

```
LIST
10 FOR I=1  TO 1000 :NEXT
```

このNEXTの部分を20行として分割するときには，まずカーソルをNEXTの前のコロンの所に持ってきて，[INS]を1回押してから20と打ち，再びカーソルを2へ戻します．

```
10 FOR I=1  TO 1000 [2]0 NEXT
                    └─カーソル
```

次に，[INS]を押し続けて，20以下を画面上の次の行へ持ってゆきます．

10 FOR I＝1 TO 1000□
　　　　　　　　　　└カーソル
20NEXT

ここで，カーソルを，1つ[↓]して，[CTRL]＋[INS]してください．すると，2つの行の間が，1行離れます．

10 FOR I＝1 TO 1000

□←カーソル

20NEXT

次に，カーソルを10の表示されている行へ持ってゆき[↵]．するとカーソルは20の表示されている行へ移動するのでここでも[↵]．これで分割はOK．再びリストを取ると，

LIST 10
10 FOR I＝1 TO 1000
LIST 20
20 NEXT

となります．

では，これをもとに戻してみましょう．まずリストを取り，カーソルを10行の表示されている行の右端に移動してください．そこでスペースを押すと，画面は次のようになります．

10 FOR I=1 TO 1000
□←カーソル
20 NEXT

ここで，[CTRL]+[DEL]をしてください．すると画面は，

10 FOR I=1 TO 1000
□ 20 NEXT
└カーソル

となります．ここで，カーソルを1000の次に持ってゆき，[DEL]を押し続けると，

10 FOR I=1 TO 1000[2]0 NEXT

となるので，もう1回[DEL]を押して，コロン(:)を押してから，[↵]します．試しにLIST 10を実行すると，

```
LIST 10
10 FOR I=1 TO 1000：NEXT
```

となります.

なお，これらの操作は，画面上の最下行近くで行うと失敗する（この例だと20行に当たる部分が画面から消えてしまう）ことがあるので，注意が必要です.

DIR命令

DIR 命令は，ディスクに登録されているファイル名を画面に表示する命令で，マイクロソフト系 BASIC の FILES に相当します.

777 BASIC におけるこの命令の特徴は，ファイルの指定にワイルドカードを使えるということで，たとえば，

DIR "A：＊．BAS"（＊．拡張子）同一拡張子の全ファイル名を表示
DIR "A：TEST．＊"（ファイル名．＊）同一ファイル名の全ファイル名を表示
DIR "A：T？？？．＊" ？の入った部分以外が共通の全ファイル名を表示

という具合に使えるわけです．また，表示されているファイル名が1行にひとつずつで，しかも左にコマンドを入れるための空白を置いて " で囲まれていますから，LOAD や RUN などのコマンドを実行するときに便利です.

```
DIR
        "A：TESTRR 1.BAS"
RUN[↵]  "A：TESTRR 2.BAS"
        "A：TESTRR 3.BAS"
```

上のように DIR を実行したあと，TESTRR 2 を実行したいときは，カーソルを目的の行へ移動して RUN [↵] とタイプすれば OK.

なお，ファイル名をプリントアウトする命令（マイクロソフト系 BASIC の LFILES に相当するもの）はありませんので，[CTRL]＋[P] のあとに，DIR を実行する必要があります.

（2）EDITとDEBUG

ここまでの説明でもわかるとおり，777 BASIC にはプログラムを自分で作る人のために，いろいろと便利な機能が備わっています．デバッグするときにリストを修正しても変数の値が初期化されないという細かな気配りもしていますが，特によく配慮されているのは EDIT と DEBUG でしょう．これらのコマンドは，デバッグの効率を高める有効なツールで，特にいつもバグを出してばかりで困っている人には役に立つシロモノ．以下に具体的な使い方を示しますので，十分に

役立ててください．

EDIT

EDIT コマンドは文字どおりエディットを行う命令で，書式は，

```
EDIT 〈行番号〉
```

となります．行番号を省略したときには，最後にエディットした行，または最後に実行された行を指定したことになり，実行するとテキスト画面はクリアされ，指定した行のみが表示されます．その行の修正が終ったら［SHIFT］+［↑］または［↓］で前後の行を呼び出し，その時点で，修正した行は自動的にメモリに記憶されます．

修正が終了したら［↲］を押すことで完了してエディットモードから抜けますし，［ESC］を押すと，エディットを中断してこのモードから抜けることもできます．また，エディットモードで［H］キーを押すと，

```
Help [B,E,L,S]:
```

と表示され，B，E，L，Sの各キーで，表5-3に示したような機能を選択使用することができ，とりわけSで得られる文字列の検索は，長いプログラムなどの変数名の変更等に威力があるものです．

コマンド	機　　能
B	Begin エディット行をプログラムの先頭に移す
E	End エディット行をプログラムの最後に移す
L	Line エディット行を変更する．変更先の行番号を入力すればよい
S	Search 探したい文字列（20文字以内）を入力すると，その文字列を含む行にエディット行が移動する．［↲］を押すごとにその文字列を含む行に次々と移る

表5-3　EDITモードのヘルプコマンド

DEBUG

DEBUG は文字どおりデバッグのためのコマンドで，書式は，

```
DEBUG 〈行番号〉
```

です．行番号を省略したときはEDITと同様で，これを実行すると画面がスクロールし，20行目にデバッグのコマンドメニューが，

Command [C D E G L P Q R S T U 1 2]:

と表示され，デバッグモードに入ります．この状態でデバッグを進めてゆくわけですが，このときスペースバーを押すと，

コマンド	入力形式	機能と使いかた
C	Clear	通常のCLEARコマンドと同じ
D	Debug[行番号]	DEBUGコマンドと同じ
E	Edit[行番号]	通常のEDITコマンドと同じ
G	Goto[開始行番号] [/ブレーク行番号 [：ブレーク行中の文番号] [(実行回数)],……]	それまでの変数の値を保持したまま，プログラムを部分的に実行する 省(行番号)　i)直前に実行されたDEBUGコマンドの指定行　ii)前のGコマンドでのブレーク行　iii)直前のSまたはTコマンドでの実行文の次の行，から開始 (文番号)　0とみなし，前の行の最後まで実行 (回数)　1回
L	List[開始行番号] [−][終了行番号]	通常のLISTコマンドと同じ．ただし，スクリーンエディタ機能は使えない
P	Print[@デバイス番号,]式，変数または定数	通常のPRINTコマンドと同じ
Q	Quit {Y / N}	Yでデバッグモード終了 Nは続行
R	Run[行番号]	通常のRUNコマンドと同じ
S	Step[実行する文の数]	指定した数の文を実行 DEBUGコマンドで特定の行を指定した直後，またはGコマンドのブレーク行やTコマンド，Sコマンド，STOP，ESCキーなどで実行か中断した直後のみ有効 省1
T	Trace[実行行数]	指定した行数分だけプログラムを実行 Sコマンドと同じ条件でのみ有効 省1
U	Undefined variable detection {Y / N}	未定義の変数を検出．エラー表示を出すときはY，必要ないときはN
{1 / 2}	Print式	文を1つ実行するごとに，指定した式を繰り返し実行 1では結果を24行目に2では25行目に表示

[ ]部分は省略可，{ }はどちらか一方を選択

表5-4 DEBUGコマンドのコマンド表

(Direct Mode)

と表示され，表示画面上で1行以内のコマンド（BASICのコマンド）入力や，新たに行をプログラムに追加することができます．

デバッグコマンドは表5－4に示したとおりですが，いくつか補足を加えておくことにしましょう．

1及び2のコマンドについては，コマンドメニューからこれらを選択すると，

1st print items (2nd print items)
Print

と表示されますが，ここでチェックしたい変数名等を入力すると，画面上の24行（または25行目）に，R，G，T，S等のコマンドで，文をひとつ実行するごとに指定した変数の値を表示し，その変化の様子を知らせてよこします．まァこれが普通の使い方なのですが（これだけでもえらく便利），Printと表示されたところでBSなどを用いてPrintの文字を消してしまい，任意の命令を入力するとそのまま実行することができるのです．たとえばデバッグコマンド1で通常のprint N（Nは変数名）を指定し，次いで2で，

PAUSE 5

と指定したとします．PAUSE Nは，約N/10秒間プログラムの実行を中断する命令です．こうすると，プログラム中のひとつの命令を実行後，チェックしたい変数の値を表示し，2で少し時間をつぶしてから次のステップに進むようになります．変化の大きい変数などのチェックには，このような方法を用いるとプログラムの流れが遅くなり，変数の値の1回の表示が長くなってチェックし易いという利点が生まれます．

通常，値の代入されていない変数は整数型，実数型では0，文字型ではヌルストリングとして処理されるのに対し，Uコマンドを実行するとR，G，T，S等のコマンドで，文を実行中に値の代入されていない変数が式の右辺に現れた場合にエラー表示を行い，実行を中断させるようになります．

このほか，777 BASICにあるREM DEBUGコマンドをプログラムに入れておくと，デバッグモード中のみに実行されるステートメントを書くことができます．たとえば，

REM DEBUG PRINT A , B

という命令をプログラムに入れておくと，通常のRUNコマンドでの実行中はリマーク（注釈）と見なされるのですが，デバッグモード中ではREM DEBUGのあとに続くPRINT A，Bという命令が実行され，変数A，Bの値を表示してくれるのです．

（3）特殊な関数

777 BASICで使用できる関数のうち特徴的なものに，COND，PROG，SETQがあります．

CONDの書式は，

COND（条件式，式1，式2）

で，条件式の値が真（0以外）なら式1，偽（0）ならば式2をその値とします．たとえば，

COND（a＜0，－a，a）

とすると，aの絶対値がその値となります．

PROGの書式は，

PROG（式1，式2，……，式n）

で，この関数は式を1から順番に評価し，最後の式の結果をその値とします．

SETQの書式は，

SETQ（変数，式）

で，式を評価し，結果を変数に代入し，さらにこの式の結果を値とします．たとえば，

SETQ（B，10）

とすれば，変数Bに10が代入されると共に，この関数の値も10となります．

このような働きを持つ関数ですが，DEF FN 命令（ユーザーが関数を定義する命令）中に使用することで，再帰的な呼び出しをすることができるようになります．その例が次のページのリスト5-1で，これはある数（ここでは行番号20で変数にキー入力された数値）の階乗を求めるプログラムで，行番号10でそれを実行するファンクションの定義を行っています．

この行をよく見ると，定義しようとする関数自身をその右の式で使用しており，このような定義の方法を，Logoでも紹介したように再帰呼出しと呼びます．

このプログラムを見れば，

$$f(n) = \begin{cases} 1 & [n=0] \\ n \times f(n-1) & [n>0] \end{cases}$$

という，階乗の再帰的な定義をそのままコーディングしてあることは，すぐにわかるでしょう．

BASICでこのように関数を再帰的に定義できるということは，実に珍しく注目に値するものと考えられます．しかし，再帰呼出しを用いてゆくには少々の慣れが必要で，この短いプログラ

ムを手本にして習熟して欲しいと思います．いずれにしろ，この方法を自分のものにしてファンクションが自由に使えるようになると，プログラミングはぐんと効率的なものになるでしょう．

List 5-1

```
10 DEF FNFACT(N)=COND(N=0,1,N*FNFACT(N-1))
20 INPUT X
30 PRINT X;"ﾉ ｶｲｼﾞｮｳﾊ";FNFACT(X)
```

```
Run
?16
16ﾉ ｶｲｼﾞｮｳﾊ20922789888000

* End *
Ready
```

リスト5-1の実行例

（4）リンクモジュール

SMC-777のオリジナルディスクには，"COPY．PAC"，"PIC．PAC"，"EFFECT．PAC"の3つのリンクモジュールファイルが入っていて，これらをLINKというコマンドで，メモリにロードすることができます．この3つにはそれぞれ，画面のハードコピーを取るためのコマンド，画像データのロードとセーブに関係するコマンド，効果音を出すコマンド群が入っています．

たとえば，

```
LINK "EFFECT.PAC" ↵
```

としたあと（拡張子を省略することも可能），アンダーライン（ _ ）を使って，

```
_UP 1 (100) ↵
```

を実行すると，上昇音（上に昇っていく感じの効果音）が，SMC-777のスピーカーから出てきます．UP 1というのは，EFFECT．PACに収められている拡張コマンドで，カッコのなかの数値で効果音の長さが決まります．拡張コマンドを実行するには，このほかにCALLがあって，

```
CALL UP 1 (100) ↵
```

としても，同じ結果になります．

このEFFECT．PACを利用した簡単なプログラムを次に示します．

List 5-2

```
10 WHILE 1
20   K$=INKEY$(1)
30   IF K$="U" OR K$="u" THEN CALL UP1(IRND(256))
40   IF K$="D" OR K$="d" THEN CALL DOWN1(IRND(256))
50 WEND
```

EFFECT．PACをLINKしたあとこのプログラムを実行すると，Uのキーを押すと上昇音，Dのキーを押すと下降音が出るようになります（プログラムを止めるには ESC を押します．忘れないように！）．

ところで，LINKコマンドはプログラム中では実行できない（実行しようとすると，プログラムが中断されてしまう）ので，プログラムを作ったときには，実行前にいちいち，

```
LINK "EFFECT.PAC"
```

を実行する必要があります．ただし，一度作ってしまったプログラムはSAVEするときに／Cを付けることで，このメンドウな問題を解決できます．試しにこのプログラムをセーブしてみましょう．

```
SAVE/C "UPDOWN" ↵
```

と入力すると，

```
Enter Startup Command Line (s)
```

と表示されますので，ここで，

```
LINK "EFFECT.PAC" ↵
```

としてください．これで，このUPDOWNというプログラムをロードすると，自動的にEFFECT．PACがリンクされます．

（5）ファイルのロードとセーブ

777 BASICはSony Filerでの1つのコマンドですが，一度BASICに入ってしまえば，ディスク上のファイルの操作は，大部分BASICを抜け出すことなく行えます．マイクロソフト系のFILESに相当するDIRは，さきほど説明したとおりの機能を持っていますが，プログラムのセー

ブやロードを行う命令も当然存在して，SAVE，LOADという名前になっています．

SAVE

セーブのためのコマンド，SAVEの書式は，

```
SAVE [〈／A または ／P〉] [〈／C〉]  "[〈デバイス名：〉] 〈ファイル名〉 [ .〈拡張子〉]"
          [,〈機械語先頭番地〉,〈最終番地〉 ]
```

とかなり長いものになっています．ただしほとんどの部分は省略可能で，たとえばTEST .BASというファイル名でBASICプログラムをセーブするには，

```
SAVE "TEST" ↵
```

でOKです．では，ほかにゴチャゴチャと付いているものに意味がないかというとそうではなく，まず，

```
SAVE／A "TEST" ↵
```

とすると，プログラムはアスキー形式でセーブされます．アスキー形式ということは，プログラムの文字のイメージそのままでプログラムがセーブされるということです(これを指定しないと，マシンの内部で処理し易い形でセーブされます)．

アスキー形式でセーブした場合の利点は，まずTYPEというコマンドで，プログラムの内容を見ることができること．TYPEはSony Filerのコマンドにもありますが，BASICでも，

```
TYPE "[〈デバイス名：〉] 〈ファイル名〉.〈拡張子〉"
```

の形で実行できます．つまり，

```
TYPE "TEST . BAS"
```

とすれば，ディスク上のファイルの内容が画面に表示されるのです．また，プログラムの結合コマンドであるMERGEを実行するときも，／Aを付けてセーブしておかなくてはなりません(MERGEについては後述)．

次に／Cですが，これは，前の項で説明したとおり，プログラムをロードしたときに自動的に実行されるコマンドを定義する場合に付けます．

なお／Pを付けると，プロテクトといって，セーブしたプログラムは，LOADはできてもLISTを取ったりすることができなくなります．これは他人にプログラムを見せなくするための方法ですが，一度設定すると二度と解除できないので注意が必要です．

デバイス名については，省略すると現在使用しているディスクが想定されますが，もちろん，

SAVE "A : TEST" ↵

とか，

SAVE "B : TEST" ↵

として，ディスクドライブの指定をすることができます．もし，増設フロッピーディスクを接続している場合に，デバイス名を省略したときにBドライブが指定されるようにしたいときは，LOGINというコマンドが用意されていて，

LOGIN "B : " ↵

とすれば変更されます．

なお，このデバイス名には，カセットやCRT，プリンタなどを指定することができて，それぞれ，

SAVE "CTR : TEST" ↵
SAVE "CON : TEST" ↵
SAVE "LST : TEST" ↵

という具合に指定します．

SAVE，LOADでは，拡張子（マニュアルではタイプ名）を省略すると自動的にBASICを表す.BASが設定されます．先にDIRの項で説明したワイルドカードを有効に利用しようとするなら，自分で決めた拡張子を設けるのも自由ですが，Sony Filerが自動的に設定したり，特別に解釈したりする名前があり，これを次に挙げますので，BASICのプログラムをセーブするときには，絶対に使わないようにしましょう．

ANN, PAC, PIC, LST, TST, EXE, MEM, LOG, IDX

最後に，機械語の先頭番地と最終番地の指定ですが，これは，BASICのプログラムと一緒に機械語のプログラムをセーブするときに指定します．ただし，この機械語はデバイス名にディスクを指定した場合には使用できないので，そういう使い方もできるのだな，と覚えておく程度でよいのではないかと思います．

DISK RESET

BASICを実行している途中，システムメニューへ抜け出さないでディスクの入れ替えを行うと，そのディスクは自動的に読み出し専用になり，書き込みができなくなってしまうことに注意する必要があります．こんな場合，書き込みをしようと思ったら，

```
DISK RESET "〈ドライブ名：〉"
```

を行う必要があるのです．このときドライブ名を省略すると，使用中のドライブが設定されます．

LOAD

セーブしたファイルをロードするコマンドがLOADです．LOADの書式は，

```
LOAD [〈／V〉] "[〈デバイス名：〉] 〈ファイル名〉[.〈拡張子〉]"
```

となります．／Vを付けた場合は，ベリファイといって，現在メモリにあるプログラムと，指定したファイルの内容が同じであるか照合します．ディスクを使用する場合，カセットと違って書き込みミスはそれほど起こりませんが，これを使って一応照合しておくと安心です．

なお，DIRを使ったあと，カーソルを移動して，表示されているファイルをロードするようにすれば，簡単にファイルをロードできるのは，DIRの項でも書いたとおりです．

（6）その他の主な特徴

DEF FONTとFONT RESET

SMC-777のキャラクタ・ジェネレータは，すべてプログラマブルですから，ユーザーが自由に定義することができます．777 BASICではDEF FONT命令がそれを行い，書式は，

```
DEF FONT 〈キャラクタコード〉, "〈フォントデータ〉"
```

になります．図5-1に示したフォントをキャラクタコード65に定義するには，

```
DEF FONT 65,"02060A123E428200"
```

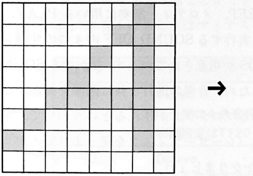
となり，定義後，

```
PRIINT CHR$(65)
```

を実行すると，図5-1のキャラクタが表示されます．一度定義されたキャラクタは，DEF FONT命令で再定義するか，リセットボタンもしくは電源を切る以外には，FONT RESET命令で初期設定に戻すまで保持されています．キャラクタの定義は，

```
FONT RESET [〈フォントセット番号（0または1）〉]
```

で，フォントセットの指定（省略時は0）により，APPENDIXのキャラクタフォント表に示したとおりの設定になります．従ってフォント表示モードを1に指定するとひらがな表示ができるようになります．



点を打つドットは1
打たないドットは0

0	0	0	0	0	0	1	0	………… 02
0	0	0	0	0	1	1	0	………… 06
0	0	0	0	1	0	1	0	………… 0A
0	0	0	1	0	0	1	0	………… 12
0	0	1	1	1	1	1	0	………… 3E
0	1	0	0	0	0	1	0	………… 42
1	0	0	0	0	0	1	0	………… 82
0	0	0	0	0	0	0	0	………… 00

図5-1 DEF FONT

実数型

SMC-777の実数型（浮動小数点型）の変数は8バイトで，仮数部が7バイトのBCD表現，残りの1バイトが指数及び符号となっています．777 BASICでは，特に型宣言を行わない限り，変数はこの実数型になります．

この実数型を用いると，計算機内部表現（2進）と10進数の間の誤差に悩まされることもなく，計算精度も上がる反面，演算速度は遅くなるという結果を生みます．

ラベル

GOTOやGOSUBの飛び先を，ラベルを使って指定することができます．この機能はN$_{88}$-BASICなどにも備わっていますが，これを使うことで飛び先の機能などを名前で示すことができ，何よりリナンバーをすればコロコロ変ってしまう行番号という無味乾燥なシロモノから解放されることになります．ラベルは行の先頭にアスタリスク（*）を付けて定義しますが，具体的な使い方は，以後のプログラムにたくさん出てきますので，それを参考にしてください．

5-2 効果音とサウンド

パソコンにおけるサウンド機能の充実は，このところ天井知らず（？）に伸びてゆく気配があるようで，なかにはそれを売り物にする機種まで出る有様．市場がそんな状態なものですから，SMC-777もそれなりに充実したサウンド機能を持って世に送り出されました．通常のサウンド出力に加え，PSG (Programmable Sound Generator)というサウンド専用LSI（SN76489AN）が実装され，ゲームの効果音等に威力を発揮するほか，自動演奏を実行させたりエレクトーンとして使ったりと，様々な使途を生み出せるように作られています．この節では，BASICが備えているサウンド用のコマンドの解説と，サブルーチンに用意されている効果音を利用して，サウンド機能をもりたてるあれこれを紹介してゆきます．

(1) コマンドと効果音

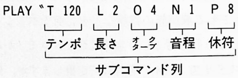
777 BASICには"ピッ"という短い音を鳴らすBEEP，メロディー演奏に用いるPLAY，和音を奏でるSOUND，またSOUNDによる演奏中止を実行するSOUND OFFの4つのサウンド用基本コマンドのほか，演奏終了確認のための関数EOSが用意されています．なかでもSOUNDコマンドは，マルチタスク処理方式が採用されているため，音楽の進行と共に画面を動かすことが可能になります．ですから目と耳を同時に楽しませるプログラムが作れるというわけで，案外カラオケプログラムなどに効果があるかもしれません（レーザーディスクを使うより安あがりだし…）．まずは，その基本となるコマンドの説明からかかりましょう．

BEEP

BEEPコマンドは，単に"ピッ"という音を出すだけのものですが，プログラムによる処理の経過を音で示したり，また警告などに利用すると役に立つ存在です．さらにFOR～NEXTを使用して音を連続的に出すと，これはこれで結構使い道があるものです．

PLAY

PLAYコマンドは，サブコマンドを以降に続けて入力することで，メロディを演奏します．表5-5に示すのがサブコマンドの一覧で，テンポ(Tempo)，オクターブ(Octave)，長さ(Length)，音程（Note)，休符（Pause）等から成っており，書式は，



というように，任意のサブコマンドをいくつか並べてサブコマンド列として与えます．

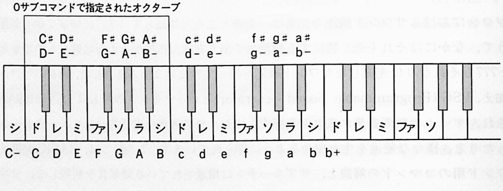
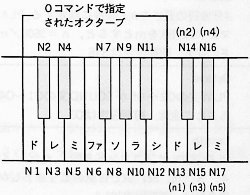


図5-2 Nサブコマンドの代りにCDE…で音程を指定

音程については，さらに図5-2に示すように，Nコマンドの代わりに，CDEF……gabといった形に，2オクターブ分を指定することができます．なお，半音上げるときは，プラス（+）またはシャープ（#），半音下げるときはマイナス（−）を音程を表すサブコマンドのあとに付けます．

<table>
<tr><th>コマンド</th><th>説　　明</th></tr>
<tr><td>T</td><td>Tempo
1分間に4分音符を何回数えるかを指定する
初期設定はT120</td></tr>
<tr><td>O</td><td>Octave
1～5の数字でオクターブを指定する．初期設定はO3</td></tr>
<tr><td>L</td><td>Length
音の長さを全音符の何分の1かで指定する
付点音符はピリオドで表される
例：付点8分音符　＝L8.
初期設定はL4</td></tr>
<tr><td>N</td><td>Note
指定されているオクターブのドの音を1として与えられた音階を演奏する．小文字nで指定すると，1オクターブ上の音になる



</td></tr>
<tr><td>P</td><td>Pause
Lサブコマンドと同様の形式で与えられた長さだけ休みをとる
例：付点8分休符　＝P8.
数字を省略するとP4とみなされる</td></tr>
<tr><td>〈　〉</td><td>囲まれたサブコマンドを，最初に書かれた数だけ繰り返して実行する
例："〈2N1P4〉"="N1P4N1P4"</td></tr>
<tr><td>^G</td><td>グラフィックサブコマンドへの移行
これ以降の文字列をDRAW文のサブコマンドとして実行する</td></tr>
</table>

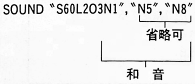
表5-5　PLAYのサブコマンド

また，このように音程をアルファベットで指定する場合，Lサブコマンドを用いずに，アルファベットのあとに音の長さを表す数字を付けて指定することができます．

小学生のころ音楽の授業をサボッたむきは，何かひとつ楽譜を睨んで，これらのサブコマンドと対比させてみるとよいでしょう…….

SOUND

SOUND コマンドも，PLAY コマンド同様サブコマンドによって，メロディ及び和音を演奏するものです．PLAY コマンドとのサブコマンドの違いは表 5 - 6 に示したとおりで，記述方法も PLAY とほぼ同じなのですが，音程の部分のみ注意が必要です．



コマンド	説　　明
S	Speed 4分音符の長さをn／60秒に指定する．PLAY文のTサブコマンドの引数をmとすると，n＝3600／mのとき同じ速さになる．初期設定はS16
O	Octave PLAY文のO2～O5がSOUND文のO1～O4に対応していることに注意．初期設定はO3
L	Length PLAY文のLサブコマンドと同じ．ただし，引数には 1，2，4，8，16，32，64と付点のピリオドしか用いることができない．初期設定はL4
N	Note PLAY文のNサブコマンドと同じ
P	Pause PLAY文のPサブコマンドと同じ．ただし，引数には 1，2，4，8，16，32，64と付点のピリオドしか用いることができない
〈　〉	PLAY文のものと同じ

表5-6　SOUNDのサブコマンド

SOUND OFF

SOUNDの実行と同時にほかのコマンドを実行している際，ほかのコマンドの実行終了に合わせ，SOUNDの実行が途中でも強制的に演奏を中止させるコマンドです．

EOS

SOUND命令では，マルチタスク的に音とほかの処理が同時に行われるのは前に書いたとおりですが，EOSは，その音が終ったかどうかを判断する関数で，引数はありません．

SOUNDによる音が鳴っていないときは−1，そうでないときは0を値とします．

効果音

オリジナルディスクに収められているリンクモジュール，EFFECT．PACには，PSGを利用して効果音を発生させる拡張コマンドが用意されています．これらは前節で説明したとおり，LINKでリンクしたあとアンダーバー（＿）もしくはCALLで呼び出すことが可能です．このEFFECT．PACに内蔵されているサブルーチンは以下のとおり．

・上昇イメージ音：UP 1 (n)，UP 2 (n)，UP 3 (n)
・下降イメージ音：DOWN 1 (n)，DOWN 2 (n)，DOWN 3 (n)
・車の音：CAR(n)
・ガンの音：GUN(n)
・雑音：NOISE(n)
・パニック音：PANIC(n)

引数は0～255の整数で，値が大きくなるほど変化が遅くなります．また，引数の記入を省略した場合は自動的に128が設定されるようになっています．これらの効果音の組合せは，文字どおりゲームプログラミングに大きな効果を生むものと思われます．

（2）PLAYプログラミング

PLAYコマンドに慣れるため，とりあえず自動演奏プログラムを作ってみることにしましょう．PLAYの基本は楽譜に対応した文字列のサブコマンドを入れてやればよいわけで，はっきり言えばルーチンワークのようなもので，難かしくも何ともありません(根気は要りますが)．ここでは1曲のみではおもしろくないので，3曲から選択できるようにプログラムを書きました．曲名は「青春時代」，「秘密の花園」，「Yesterday Once More」．選曲が趣味に合わないなどと文句をつけるむきには，自分でやれ，とあらかじめお断りしておきましょう（そうやっているうちにパソコンと気持ちが通じ合うようになるのです）．次のページのリスト5-3を見てください．

行番号40～130までは曲を選択するためのルーチンでして，曲のデータ（サブコマンドの文字列）はDATA文を用いて270～690行に記憶させています．自動演奏のためのメインプログラム

は150〜190行の5ステップで，この間のループで，1小節のデータを読んでは PLAY で鳴らしています．

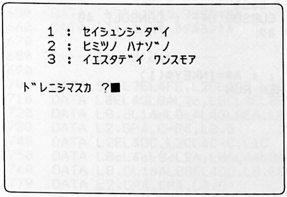
少し気を付けなくてはならないのは，繰り返しの終了を DATA 文の END（または end）というデータで判定しているところぐらいです．このように簡単な手順で，自動演奏を実行することができるのです．

List 5-3

```
 10 '
 20 ' --- PLAY DEMO ---
 30 '
 40  WIPE
 50  PRINT "     1 : ｾｲｼｭﾝｼﾞﾀﾞｲ"
 60  PRINT "     2 : ﾋﾐﾂﾉ ﾊﾅｿﾞﾉ"
 70  PRINT "     3 : ｲｴｽﾀﾃﾞｲ ﾜﾝｽﾓｱ"
 80  PRINT
 90  INPUT " ﾄﾞﾚﾆｼﾏｽｶ ";MSL
100  IF MSL=1 THEN RESTORE 280 : GOTO *LOOP
110  IF MSL=2 THEN RESTORE 450 : GOTO *LOOP
120  IF MSL=3 THEN RESTORE 530 : GOTO *LOOP
130  END
140 '
150 *LOOP
160  READ S1$
170  IF S1$="END" OR S1$="end" THEN *S_END
180  PLAY S1$
190  GOTO *LOOP
200 '
210 *S_END
220  PRINT "END"
230  INPUT "RESTART (Y/N)" ;A$
240  IF A$="Y" OR A$="y" THEN RUN
250  END
260 '
270 ' -- Sound data --
280 ' -- ｾｲｼｭﾝｼﾞﾀﾞｲ --
290  DATA L8eeL4.eL8f#L4dL8BBL2B
300  DATA P4L8ABL4GL8ABL2B
310  DATA P4L8ccL4.cL8eL4BL8GBL2B
320  DATA P4L8f#f#L4f#L8ef#L2f#
330  DATA P4L8eeL4.eL8f#L4dL8BBL2B
340  DATA P4L8ABL4GL8ABL2B
350  DATA P4L4cL4.cL8eBBGBL2B
360  DATA P4L8ABL4GL8F#EL1E
370  DATA L4eeL8eecAL2A
380  DATA L4gf#eL8BGL2G
390  DATA P4L8ABcL4.eP4L8BBeL4.g
400  DATA P4L8f#gL4f#L8ef#L1f#
410  DATA L4ggL8ggf#eL1e
420  DATA L4f#edL8ABL1B
430  DATA P4L4EGABBgeP4L8f#eL4dL8f#eL1e
440  DATA end
450 ' -- ﾋﾐﾂﾉ ﾊﾅｿﾞﾉ --
460  DATA T12002L8ggfeeP8eeededc4.P8ggfeeP8eeefged4.P8
```

```
470  DATA ddcBB4Baagfe4a4.aaaf#f#4f#g#g#aag#4e4P2
480  DATA eeee4.Pf-4ed4c4.<2d4gg4.P>cBcd4cde4def4ef4a2f2e4g4
490  DATA g4a4b4.g4.Pg4ab4gedc2.PL4fgecde8G4.PfgL8edce4d4.
500  DATA g4a4b4.g4.Pg4ab4gedc2ca4T96g2T120ab4T96b#2T120bgeT96
510  DATA a2T120gecT69f2T120Pe4c4dT96c2T120P2
520  DATA END
530 ' -- イエスタデ゛イ ワンスモア --
540  DATA E8E8F#8G#4G#8B8B8G#8B8G#8c#8B4G#8G#4G#8B8c#4d#4G#8B4
550  DATA c#8c#2.
560  DATA G#8B8c#4g#4f#8e4d#8d#4.B8G#8B4G#8F#1
570  DATA P2P8E8E8F#8G#8B4B8B8G#8B8G#8c#8B4G#8G#4G#8B8c#4d#4
580  DATA G#8B4c#8c#2
590  DATA d#4f#4e8d#8c#8c#8c#8d#8e8d#8e8d#4c#8c#4c#8d#8e4e4c#8
600  DATA e4f#8f#2
610  DATA e4f#4g#8g#8g#8g#4f#8e8d#8e8d#8c#8c#4G#8B8B1
620  DATA P2P4e8f#8g#8g#8g#8g#8g#4f#8e8d#8e8d#8c#8c#4G#8B8B1
630  DATA P2P4c#8d#8e8d#8e8f#8f#4e8d#8e8d#8e8f#8f#4e8f#8g#4g#4
640  DATA f#8e4c#8c#4G#4G#8c#8G#8B8B1
650  DATA P2P8G#8G#8F#8E8F#8G#2G#8A4F#8F#2
660  DATA e4f#4g#8g#8g#8g#8g#4f#8e8d#8e8d#8c#8c#4G#8B8B1P2
670  DATA e4f#4g#8g#8g#8g#8g#4f#8e8d#8e8d#8c#8c#4G#8B8B1P2
680  DATA G#8G#8F#8E8F#8G#1
690  DATA END
```

© 1973 ALMO MUSIC CORP. & HAMMER AND MAILS MUSIC. THE RIGHT FOR JAPAN ASAIND TO PASIFIC MUSIC PUBLISHING CO., LTD.



PLAY DEMOメニュー画面
「青春時代」作曲 森田公一
「秘密の花園」作曲 呉田軽穂
「イエスタデイ ワンスモア」作曲 カーペンターズ

(3) SOUNDプログラミング

SOUNDを使うと和音が利用できるため，PLAYを用いた場合より，はるかに音質の豊かな演奏を実行させることができます．SOUNDの基本はPLAYとたいして変りませんので，ここでは単に曲を演奏するだけではなく，グラフィック画面上に曲の内容に伴った変化を付け，ついでに歌詞も曲の進行に合わせて画面に表示してみます．曲名は，昔なつかしい「スキーの歌」(曲と画面を合わせるための選択だ．古いなんて文句言うな！)．リスト5-4を見てください．

このプログラムは，実行するとまず最初に山の絵を描きます．そして演奏が始まったら歌詞が画面左下に表示されます．歌詞の表示は曲の進行に従ってさきへ進み，その間ずっと音符の形をした雪が舞って，山がだんだん雪化粧して終了するものです．自動演奏の方式はPLAYと同様で，

音符データはDATA文に記憶させています．メインとなるプログラムは180～280の11ステップで，全体は，190行のWHILE 1と280行のWENDでループになっています．このループのなかでまず画面左下に歌詞を書き，続いて1小節分のメロディを読み込んでから，SOUNDでこのメロディーを鳴らし，最後にバックの色を変え，音符の形をした雪を降らせる働きを持つサブルーチン（＊CCHANG）へ飛びます．

このサブルーチンは，まずDEF PALETTEでバックの色を変え，470行から530行の間を繰り返すことで，雪を降らせます．このループを抜け出す条件にEOSが使われていて，これによりSOUNDで鳴らしているメロディーと画面の動きをうまく合わせているのです．

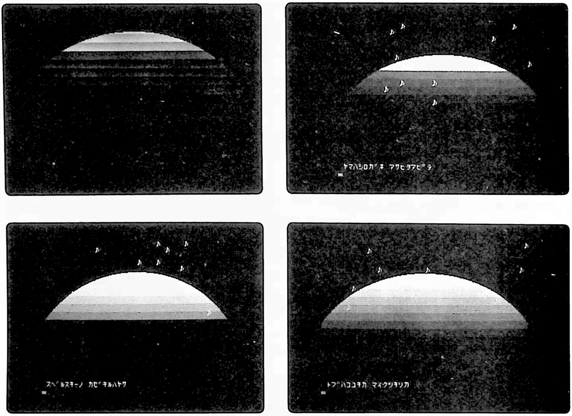
読者諸兄はすでにお気付きでしょうが，このプログラムの発展型こそ，何を隠そう絵の出るカラオケなのですよ．グラフィックス画面に楽譜を出してやるのもよし，イメージスケッチを入れるのもよし，パソコンの使い方革命だと思うのですが….

List 5-4

```
 10 '
 20 '--- SOUND DEMO ---
 30 '
 40  DEF INT A-Z
 50  DIM KASHI$(4),SNOW(9)
 60  GOSUB *KASHI ' ｶｼﾃﾞｰﾀﾉ ﾖﾐｺﾐ
 70  WIPE : CPALETTE OFF : CURSOR OFF : CONSOLE 40
 80  GOSUB *BACK ' ﾊﾞｯｸﾉｴｦ ｶｸ
 90  GOSUB *MAIN
100  PRINT "END"
110  PRINT "RESTART (Y/N)" ; : A$=INKEY$(1)
120  IF A$="Y" OR A$="y" THEN RUN
130  CONSOLE 80 : WIPE
140  END
150 '
160 ' --- Main routine ---
170 '
180 *MAIN
190  WHILE 1
200    READ S1$
210    C=C+1
220    K=INT((C-1)/4)
230    PRINT (1,22) ;KASHI$(K)
240    IF S1$="END" THEN RETURN
250    READ S2$,S3$
260    SOUND S1$,S2$,S3$
270    GOSUB *CCHANG
280  WEND
290 '
300 ' --- Subroutine  <   ﾊﾞｯｸﾉｴｦｶｸ   >  ---
310 '
320 *BACK
330  GLOCATE(0,199),1,-1
340  FOR I=1 TO 15
350    DEF PALETTE I,I/4,I,I/2
360    BOXF(0,I*10)-(319,I*10+11),I
370  NEXT
```

```
380  CIRCLE(160,0),200,160,7
390  DEF PALETTE 0,0,0,6
400  PAINT(1,150),0,7
410  RETURN
420 '
430 ' ---  <   ﾊﾞｯｸﾉｴﾉｲﾛｦ ｶｴﾙ  &  ﾕｷｦﾌﾗｾﾙ     >  ---
440 '
450 *CCHANG
460  IF C<16 THEN DEF PALETTE 16-C,16-C,16-C,16-C
470  WHILE 1
480    I=(I+1) MOD 10
490    VOUT SNOW(I),0
500    SNOW(I)=IRND(500)*2
510    VOUT SNOW(I),7
520    IF EOS THEN RETURN
530  WEND
540 '
550 ' --- Kashi data ---
560 '
570 *KASHI
580  FOR I=0 TO 3
590    READ KASHI$(I)
600  NEXT
610  RETURN
620 '
630  DATA "ﾔﾏﾊｼﾛｶﾞﾈ ｱｻﾋｦｱﾋﾞﾃ"
640  DATA "ｽﾍﾞﾙｽｷｰﾉ ｶｾﾞｷﾙﾊﾔｻ"
650  DATA "ﾄﾌﾞﾊｺﾕｷｶ ﾏｲﾀﾂｷﾘｶ "
660  DATA "ｵｫｫｺﾉﾐﾓ ｶｹﾙﾖｶｹﾙ  "
670 '
680 ' --- Sound data ---
690 '
700  DATA S25L2EL4FE,L2GL4AG,L1C
710  DATA L8EL4GL8AL2c,L8CL4EL8FL2E,L1C
720  DATA L8.dL16dL8cAL4GL8EA,L8.FL16FL8FFL4EC,L2FC
730  DATA L2.GP4,C-P4,L2.G
740  DATA L2EL4DC,L2CL4C-C,L1C
750  DATA L8cL4cL8cL2A,L8AL4AL8AL2F,L1F
760  DATA L8.GL16AL8GEL4DD,L8.EL16FL8ECL4C-C-,L2CG
770  DATA L2.CP4,CP4,L2.C
780  DATA L4.cL8dL4ec,L4.EL8FL4GE,L2FC
790  DATA L8dL4cL8AL2G,L8FL4FL8FL2E,L2FC
800  DATA L8.EL16GL8AGL4Ac,L8.CL16EL8FEL4FA,L2FD
810  DATA L2.dP4,GP4,L2.G
820  DATA L2eL4de,L2GL4GG,L2CL4GC
830  DATA L8cL4cL8cL2A,L8AL4AL8AL2F,L1F
840  DATA L8.GL16AL8GEL4Gd,L8.EL16FL8ECL4C-L8GF,L2CG
850  DATA L2.cP4,EP4,L2.C
860  DATA END
```

「スキー」作詩 時雨音羽　作曲 平井康三郎



SOUND DEMOの実行例

(4) 777オルガン

今までのプログラムがおしきせの楽譜を演奏するだけだったのに対し，ここで紹介するのは，パソコンをオルガンとして機能させ，自分で演奏を楽しむためのプログラムです．ただしこのオルガン，メロディーは出せるものの和音までは手が回らず，まァ初心者用のオルガンとでも言いましょうか．リスト5-5がそのプログラムです．REM文でプログラムの要所に解説を付けておきましたので，よく読んで内容を考えてみてください．実行すると，オルガンのキーとパソコンのキーボードとの対応が表示されて，またオクターブと何分音符かを選択できるようにしてあるため，演奏前に任意のキーで指定するようにしてください．そして，実際の演奏は(CAP LOCKをはずすことをお忘れなく，小文字ですよ)，パソコン側のキーに対応する鍵盤の位置に赤いマークが現れ，そこはかとなくオルガン演奏の味を出すように工夫してみました．なおこのプログラムにあらかじめ曲のデータを入れておき対応キーを演奏に合わせてマーキングすると，昔なつかしい自動オルガン(昔はあったんだ，昔は……)風になり，おもしろいのではないかと思います．

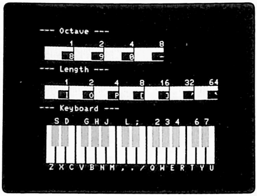
サウンド機能に関する説明はこれくらいにしておきますが，ハードウェアのマニュアルにPSGに関して詳しく解説されているため，BASICのコマンド以上の機能を望む方は，それらを参考にしてアセンブラに挑戦していただきたいと思います．また，余談になりますが本体内蔵のスピー

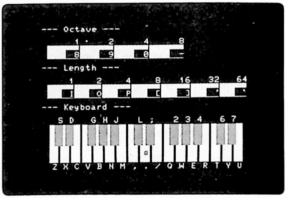
カーは，本体下部に付けられているため，机の上などで使うときには，スペーサーをかまして少々浮かすと少々音量が上がります．もちろん，より音量を必要とする場合は，アンプとスピーカーを接続するべきです(なお，ここで用いたグラフィックスコマンドに関しては，次章のグラフィックスコマンドとパレットの節を参照してください)．

List 5-5

```
 10 '
 20 ' --- PLAY DEMO < ｵﾙｶﾞﾝ > ---
 30 '
 40  CONSOLE 40 : CURSOR OFF : FONT RESET 0
 50  WIPE
 60  DIM KP(1,30)
 70  GLOCATE (0,180),1,-1
 80  GOSUB *KPOINT_DATA_IN                    ' ﾎﾟｲﾝﾄﾉ ｲﾁﾉ ﾃﾞｰﾀｦ ｲﾚﾙ ﾙｰﾁﾝﾍ
 90  GOSUB *KBOARD                            ' ｷｰﾎﾞｰﾄﾞｦ ﾂｸﾙ ﾙｰﾁﾝﾍ
100  CCOLOR 4                                 ' ﾎﾟｲﾝﾄﾉ ｲﾛｦ ｱｶﾆｽﾙ
110 '
120 *LOOP                                     ' ﾒｲﾝﾙｰﾁﾝ
130    A$=INKEY$(2)
140    IF A$="" THEN 130
150    N=INSTR("zsxdcvgbhnjm,l.;/q2w3e4rt6y7u890-iop[]'`",A$)
160                                           ' ﾅﾆｦ ｵｼﾀｶ ﾐﾙ
170    IF N=0 THEN 130
180    IF N>29 THEN *MODSET                   ' ﾓｰﾄﾞｾｯﾄﾅﾗ ﾓｰﾄﾞｾｯﾄﾙｰﾁﾝﾍ
190    IF O=4 AND N>24 THEN 130
200    PRINT (KP(0,N),KP(1,N));CHR$(236);' ﾎﾟｲﾝﾄｦ ｳﾂ
210    PLAY"N(N)"
220    PRINT CHR$(8);" ";                     ' ﾎﾟｲﾝﾄｦ ｹｽ
230  GOTO *LOOP
240 '
250 *MODSET
260  N=N-29
270  IF N>4 THEN *TENPO                       ' ﾃﾝﾎﾟﾉ ｾｯﾃｲﾅﾗ *TENPO ﾍ
280  PLAY "O(N)"                              ' ｵﾝｶｲﾉ ｾｯﾃｲ
290  O=N
300  GOTO *LOOP
310 '
320 *TENPO                                    ' ﾃﾝﾎﾟﾉ ｾｯﾃｲ
330  N=2^(N-5)
340  PLAY "L(N)"
350  GOTO *LOOP
360 '
370 *KBOARD                                   ' ｷｰﾎﾞｰﾄﾞｦ ﾂｸﾙ
380  FOR I=1 TO 17
390    BOXF(16*I+4,6)-(16*I+16,51),7
400  NEXT
410  FOR I=1 TO 17
420    IF (I MOD 7)=3 THEN 450
430    IF (I MOD 7)=0 THEN 450
440    BOXF(16*I+11,26)-(16*I+23,51),15
450  NEXT
460  PRINT (3,22);"Z X C V B N M , . / Q W E R T Y U"
470  PRINT (4,15);"S D   G H J   L ;   2 3 4   6 7"
```

```
480  FOR I=0 TO 6
490    BOXF(40*I+16,84)-(40*I+54,100),6
500  NEXT
510  FOR I=0 TO 6
520    BOXF(40*I+38,85)-(40*I+54,94),0
530  NEXT
540  FOR I=0 TO 3
550    BOXF(48*I+16,132)-(48*I+62,148),6
560  NEXT
570  FOR I=0 TO 3
580    BOXF(48*I+44,133)-(48*I+62,142),0
590  NEXT
600  PRINT (0,1);" --- Octave ---"
610  PRINT (4,3);"   1      2      4      8"
620  PRINT (4,5);"   8      9      0      -"
630  PRINT (0,7);" --- Length ---"
640  PRINT (4,9);"  1    2    4    8   16   32   64"
650  PRINT (4,11);"  I    O    P    [    ]    '    `"
660    PRINT (0,13);" --- Keyboard ---"
670  RETURN
680 '
690 *KPOINT_DATA_IN
700  RESTORE 770
710  FOR I=1 TO 29
720    READ KP(0,I),KP(1,I)
730  NEXT
740  RETURN
750 '
760 *KPOINT_DATA
770  DATA  3,20, 4,18, 5,20, 6,18, 7,20
780  DATA  9,20,10,18,11,20,12,18,13,20,14,18,15,20
790  DATA 17,20,18,18,19,20,20,18,21,20
800  DATA 23,20,24,18,25,20,26,18,27,20,28,18,29,20
810  DATA 31,20,32,18,33,20,34,18,35,20
```





PLAY DEMO〈オルガン〉の実行例

5-3 ディスク関連のコマンド

今までにも何度か書いてきたように，SMC-777はマイクロフロッピーディスクのドライブを標準装備したパソコンであり，この777 BASICもディスクの能力を十分に活かせるようになっています．

本節では，ここまでに説明したDIR，LOAD，SAVEといったコマンド以外のディスク関連コマンドを紹介してゆきます．本節を読みこなすことで，せっかくのディスクを宝の持ち腐れとしない，賢い777オーナーとなってください．

(1) OPEN／CLOSE

すでに何度も書いたように，ディスクには，ファイルという形で情報が詰め込まれています．ファイルにはデータを格納する形の違いにより，アスキーファイルとバイナリーファイルの2種類があり，たとえばBASICのプログラムをセーブするときに／Aを付けるとアスキーファイル（アスキー形式のファイル）としてセーブされます．

ファイルをデータとして扱う場合には，これを読んだり書いたりするときに，コンピュータとそのファイルの間に橋をかけて，情報を受け渡しできるようにしなければなりません．

たとえばディスクのなかに〝TEST 1〃，〝TEST 2〃，〝TEST 3〃という3つのアスキーファイルがあったとき，OPEN／I　#1，〝TEST 1〃を実行すると，SMC-777とTEST 1の間には図5-3のように#1という橋が（入力用に）かけられます．同様にOPEN／Oでは出力用の橋がかけられます．

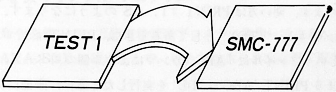


図5-3　ファイルから橋をかける

OPEN／R，OPEN／Sは内部表現形式のデータ（バイナリーファイル）を読み書きするための橋をかける命令で，内部表現形式とはBASICのプログラムがメモリに格納されている状態のように，マシン内部でのデータ表現と同じ形式のことです．バイナリーファイルはアスキーファイルと違い，直接TYPEすることはできませんが，スペースの効率がよく，マシン内部での処理効率もよいのでよく使われます．

OPEN／Rではバイナリーファイルの入力用に橋がかけられ，OPEN／Sではバイナリーファイルの出力用に橋がかけられます．

ところで，すでに橋がかけられて（OPEN されて）いるファイルに，もう1本橋をかけようとするとおこられてしまいます．これは，同じファイルには2本以上の橋をかけられないためで，いらなくなった橋を切るにはCLOSEを使います．

```
OPEN／I #1，"TEST 2"
OPEN／O #1，"TEST 2"
```

ではまずいので，

```
OPEN／I #1，"TEST 2"
CLOSE #1
OPEN／O #1，"TEST 2"
```

のようにするわけです．また，

```
OPEN／I #1，"TEST 2"
OPEN／O #2，"TEST 4"
```

のように別のファイルには別の橋でもって対応します．なお，用事が終ったら橋は切っておく（CLOSEする）ようにしましょう．

（2）PRINT＃／INPUT＃

さて橋をかけたら，データを出し入れしなければなりません．データの出し入れのための命令は，アスキーファイルとバイナリーファイルとでは違っていて，アスキーファイルの書き込みにはPRINT #を使います．使い方はPRINT #1，A$ のようになります．

たとえばOPEN／O #3，"TEST" としてあるときに，PRINT #3，"ABC"，"DE" を実行すると "TEST" というファイルに「ABC（カンマによる5個のスペース）DE」という1行が書き込まれます．つまりPRINT "ABC"，"DE" を実行したときに，画面に文字を出すのと同じ形式で，ファイルにも書き込んでいくのです．またPRINT #に続く変数の並びが[；]または[，]で終っていなければ，区切りとして[CR LF]が書き込まれます．

書き込みが終ったらCLOSEしてください．CLOSEしないと，OPENで作った橋の上を移動中のデータが失われることになってしまいます．

アスキーファイルからデータを読み込むには，INPUT # またはINPUT／L #を使います．INPUT #は[，]または[CR LF]を，INPUT／L #は[CR LF]のみを，データの区切りとみなします．また，読み込むデータが残っているかどうかを知るのにEOFを使います．

データが通る橋に#2を使うとすると，まずOPEN ／I #2，"TEST"で#2という橋をかけてINPUT #2，A$ というように実行します．これでA$ にはTESTというアスキーファイルに入っている最初のデータが読み取られてきます．あとはINPUT #が実行されるたびに，次々と

読み込まれてきます．最後のデータを読み込むと，EOF(2)の値が－1になり，読み込みが終了したことを知ることができます．

EOF(n)のnは読み込みに使っている#nと同じ番号を指定します．IF EOF(2)=－1 THEN CLOSE : ENDなどの1行をINPUT #の後に入れておけば，「最後の行 (End of File) を読み込んだのに，まだ読み込もうとしている」とおこられなくてすみます．EOFは0と－1の値しかとらず，これが論理的な偽と真に対応するので，IF EOF(2)とかWHILE NOT EOF(2)のように使うことができます．

(3) SEND／RECEIVE

SENDとRECEIVEは，バイナリーファイルを読み書きするときに使う命令です．SENDはPRINT #と同じように，SEND #1，A$ のようにして使います．PRINT #では[,]が文字列中にあると，区切り記号となりましたが，SENDではそういうことはありません．また，PRINT #では(PRINTと同じように)数字を文字列にして書き込みますが，SENDはマシン内部の形式そのままで書き込むので，ファイルが小さくてすみます．

RECEIVEは，SENDによって書き込まれたファイルを読み込むときに使うもので，INPUT #と同じように使います．ただしアスキーファイルとは違ってEOFが使えませんので，SENDで書き込んだ数以上にRECEIVEで読み込まないように工夫しなければなりません．

BASICのプログラム(アスキーセーブされていないもの)は，このバイナリーファイルの形で登録されておりSEND，RECEIVEを使えば，こういったプログラムファイルをデータとして扱うことさえできます．

(4) データファイルを扱うサンプル・プログラム

これまでに説明した命令を使って，万能情報管理プログラムを作ってみましょう．入力した情報を次々とファイルDATA. BOXに詰め込んで，好きなときに自由な形で取り出してきてくれるプログラムです．

プログラムは3つの機能を持っています．最初にこのプログラムを使うときや，DATA. BOXを空にしてしまいたいときに使うINITIALIZE，INITIALIZEで作ったDATA. BOXにデータを詰め込むAPPEND，APPENDで詰め込んだ情報のなかから必要なものだけを取り出してくれるSEARCHです．

まずはリストを見てみましょう．

List 5-6

```
1000 '
1010 ' === general purpose data management system ===
1020 '
1030  WIPE : FONT RESET : CONSOLE 80
1040  DISK RESET
```

```
1050 DEF FNLC$(O$)=COND(O$>'Z' OR O$<'A',O$,CHR$(ASC(O$)+32))
1060 DIM A$(1000)
1070 '
1080 ' -- Main --
1090 '
1100 *MENU
1110 WIPE
1120 PRINT ,'   - GPDMS -'
1130 PRINT
1140 PRINT ,'initialize....(i)
1150 PRINT ,'append........(a)
1160 PRINT ,'search........(s)
1170 PRINT ,'quit..........(q)
1180 PRINT
1190 INPUT 'What´s your job';C$ : PRINT
1200 C$=FNLC$(C$)
1210 IF C$='' THEN *MENU
1220 IF C$='i' THEN GOSUB *INITIALIZE : GOTO *MENU
1230 IF C$='a' THEN GOSUB *APPEND.DATA : GOTO *MENU
1240 IF C$='s' THEN GOSUB *SEARCH : GOTO *MENU
1250 IF C$='q' THEN WIPE : PRINT  'job end' : END
1260 GOTO *MENU
1270 '
1280 ' -- initialize --
1290 '
1300 *INITIALIZE
1310 PRINT '[initialize mode]'
1320 INPUT 'SURE (y/n)? ',A$ : PRINT
1330 IF FNLC$(A$)<>'y' THEN RETURN
1340 PRINT '[initializing...]'
1350 OPEN /O #1,'DATA.BOX'
1360 CLOSE
1370 RETURN
1380 '
1390 ' -- append data --
1400 '
1410 *APPEND.DATA
1420 PRINT '[append mode]'
1430 OPEN /I #1,'DATA.BOX'
1440 OPEN /O #2,'TEMP.BOX'
1450 WHILE NOT EOF(1)
1460   INPUT /L #1,D$
1470   PRINT #2,D$
1480 WEND
1490 '
1500 INPUT /L 'Append data is = ',D$ : PRINT
1510 WHILE D$<>''
1520   PRINT #2,D$
1530   INPUT /L 'Next append data is = ',D$ : PRINT
1540 WEND
1550 CLOSE
1560 ERA 'DATA.BOX'
1570 REN 'TEMP.BOX' TO 'DATA.BOX'
1580 RETURN
```

```
1590 '
1600 ' -- search --
1610 '
1620 *SEARCH
1630  PRINT "[search mode]"
1640  INPUT /L "Type search string is = ",S$ : PRINT
1650  IF S$="" THEN *MENU
1660  OPEN /I #1,"DATA.BOX"
1670  WHILE NOT EOF(1)
1680    INPUT /L #1,D$
1690    IF INSTR(D$,S$)>0 THEN PRINT D$
1700  WEND
1710  PRINT
1720  CLOSE
1730  GOTO *SEARCH
```

1410〜1580行はAPPENDの部分です．1430〜1480行でDATA.BOXからTEMP.BOXへデータをコピーし，その後に入力されたデータをTEMP.BOXに書き込んでいます．そして，ここから抜けるときにDATA.BOXを削除し，TEMP.BOXのファイル名をDATA.BOXに変えています．これでDATA.BOXにデータが追加されたことになるわけです．

それではプログラムを打ち込んで実行してみましょう．

最初にMENUの画面が現れますから，まずi[↵]でinitializeしましょう．次にデータを打ち込みます．a[↵]とすると[append mode]と出て，データの入力を待っていますから，たとえば「486-7111　ascii[↵]」のように打ち込んでいきます．いくつか打ち込んだら，何も入れずに[↵]を押してもとのMENUの画面に戻してください．こんどはs[↵]でsearchを始めます．ascii[↵]と打ち込んでも，486-7111[↵]と打ち込んでも，さきほどの「486-7111　ascii」という行が出てくるのが確認できるでしょう．

SEARCHでは，入力された文字を含む行をすべて表示してくれますから，住所録などを打ちこんで郵便番号で検索すれば，同じ地区に住む人間を一目で見たりすることも簡単にできます．入力するデータはどんなものでもよく，たとえば出前を頼むときのメニューを入れておけば，どの店が何時までやっているか，かつ丼はいくらだったかなどを知ることができて便利です．

(5) MERGEとCHAIN

これまでは，データファイルについて話してきましたが，ここでちょっとプログラムファイルを扱う命令に触れておきます．プログラムファイルを扱う命令では，すでにLOADとSAVEを知っていると思いますが，この他に777 BASICには，MERGEとCHAINの2つの命令が用意されています．

MERGEは，現在メモリ上にあるプログラムと，ディスク上のアスキーセーブされたプログラムを結合させるものです．たとえばディスク上の"PROG.ASC"というプログラムをMERGE "PROG.ASC" [↵]を実行してメモリ上のプログラムと結合させると，次のようになります．

メモリ上	"PROG.ASC"		メモリ上
10 PRINT			10 PRINT
20 PRINT			20 PRINT
30 PRINT		MERGE "PROG.ASC"	30 PRINT
40 PRINT	40 WIPE	→	40 WIPE
50 PRINT	50 WIPE		50 WIPE
	60 WIPE		60 WIPE
	70 WIPE		70 WIPE

これでわかるように，MERGEによって"PROG.ASC"は空いた行番号にはそのまま，行番号が存在する場合は前のプログラムを消して，自分を書き込んでゆきます．一般にはMERGEするプログラムの行番号は，メモリ上のプログラムの行番号と重ならないように配慮する必要があります．

CHAINは，次々とプログラムを実行してゆくための命令で，2通りの使い方があります．ひとつはプログラム全体を次々と変えていく使い方で，チェインスモーカーというとタバコを切れめなくどんどん吸う人を言うのですが，そういう感じになります．もうひとつは，プログラムの一部を次々と変えていくやり方でMERGEに似ていますが，プログラムの実行がとぎれない点が違います．MERGEは，プログラム中で使ってもプログラムの実行が終ってしまいます．CHAINの使い方は，

```
CHAIN /M  /V  "PROG", 100,  DELETE 1000-2000
      ①   ②    ③      ④         ⑤
```

のような感じで，①を付けるとさきほどの2番目の（MERGEのような）使い方になります．②を付けていないと，変数に入っていた値はすべて消されてから次の④に示す行番号に実行が移ります．②を付けると，変数の値はそのままで④に示す行番号に実行が移ります．③にはCHAINするプログラム名を書きます．④はCHAINが終ったあと実行を始める行番号で，省略するとプログラムの頭から実行を始めます．⑤は，CHAINを実行する前に，もとのプログラムから消しておきたい行があるとき，DELETEに続けて消したい部分を指定します．例を挙げてみましょう．

リスト5-7のCHAINに，DELETEがないと，CHAINがリスト5-8を読み込んで実行したときにリスト5-8の102～106行が残ってしまいうまくいきません．なおCHAINで／Mを指定したときに読み込む対象となるのは，アスキーファイルだけです．

List 5-7

```
 10 '
 20 ' -- chain sample --
 30 '
 40 A=1 : B=2 : C=3
 50 CHAIN /M/V 'CCC.ASC',70,DELETE 100-
 60 '
 70 GOSUB 100
 80 END
 90 '
102 WHILE 1
104   BEEP
106 WEND
```

List 5-8

```
100 '
110 ' -- program to be chained --
120 '
130 PRINT A,B,C
140 RETURN
```

5-4 BASICによるMEMOの活用

SMC-777のハードウェアの最大の特徴がFDD 1基内蔵ということであれば，ソフトの目玉は，このMEMOであると言えましょう．少なくとも一般大衆にとって，買えばウン万円はするであろうスプレッドシート型の簡易言語が，本体を買うことでオマケに付いてくるというのは見逃せない魅力です．しかもこのMEMO，タダで付いてくるものだからたいしたことはなかろうなどと思ったら大間違い．電子黒板としての使い勝手は十分に考慮されており，書き込みや削除といったエディット関係はファンクションキーを使って簡単に行えるし，ローマ字からカタカナやひらがなへの変換も TAB キーを押すだけで可能．それでいて横が128列に縦が1999行もあり，少々どころかかなりのデータを呑み込んでしまいます．しかもFDD内蔵のメリットを目一杯生かし，簡単かつ迅速にSAVE，LOADができます．

また，検索や並べ換えといったこの類の簡易言語に付きものの機能も備えており，サブファイルを作ってファイルをより豊富なものにすることも十分可能．計算機能の方でも四則演算とべき乗計算に加え，16種類の関数が用意されているので，よほど高度な要求でも出されたとき以外，まず困ることはありません．まァ，少なくとも高い値札をぶら下げているスプレッドシート型の簡易言語のうちでも，かなりのものがこのMEMOより劣る機能しか持っていないことは間違い

なし．ユーザーの使い方次第でかなりの働きを示す，汎用タイプの電子ノートと言うことができるのです．

さて，ソニーの手先のごとくずい分とMEMOを持ち上げましたが，しかしそれでも，やはりMEMOをこのまま使ったのではおのずと限界が生じます．MEMOは作表や簡単な集計には実に便利な存在なのですが，高度なデータ処理であるとかデータを視覚化して見せるグラフ処理になったりすると，とたんにお手上げとなってしまいます．すなわち，MEMOをより積極的に使いこなそうと思ったなら，BASICなり機械語なりにリンクさせ，より豊富で強力な機能を付加してやらねばならないということです．

本節では，そんなMEMOのバージョンアップの一例として，成績のデータをMEMOで入力，集計して，しかるのちそのデータをBASICにより解析し，グラフィック処理して表示する方法を紹介してみます．なお，MEMOの基本的な操作方法は，付属のマニュアルを参照してください．

(1) 作表

まず最初に，オリジナルディスクからMEMOの部分だけを取り出して，専用のバックアップディスクを作ってください．これはオリジナルの保守という観点もありますが，むしろMEMOにあれこれデータを入れることによる記憶容量が心配なためで，フォーマット後MEMOだけをコピーしたバックアップディスクでは，おおむね250Kバイト程度のエリアが確保されます．MEMOの作表範囲が横128列，縦1999行であることを考えると（ディスプレイ表示は横3列，縦20行），これだけのフリーエリアは是非とも欲しいものです．

さて，リスト5-9を見て下さい．MEMOを用いて作成した，8人の生徒の国語，算数，理科，社会の成績リストです．このリストはデータの入力後，MEMOのソート機能を使って成績順に並び換え，その後プリントアウトしたもので，MEMOではここまでの作業を，データ入力以外はすべて5つのファンクションキーで行えます．

List 5-9

ﾅﾏｴ	ﾊﾞﾝｺﾞｳ	ｺｸｺﾞ	ｻﾝｽｳ	ﾘｶ	ｼｬｶｲ	ｺﾞｳｹｲ	ﾍｲｷﾝ
ﾕｳｺ	1	100	12	11	83		
ｱｲｺ	2	80	34	22	42		
ﾘｮｳｺ	3	70	56	33	47		
ｹｲｺ	4	60	78	44	74		
ﾏﾁｺ	5	50	90	55	58		
ｶｽﾞﾐ	6	40	15	66	67		
ﾋﾛｺ	7	30	26	77	65		
ﾏﾕﾐ	8	20	37	88	43		
ｺﾞｳｹｲ		450	348	396	479	1673	
ﾍｲｷﾝ		56.25	43.5	49.5	59.875		52.281

(2) MEMOが作るファイル

MEMOを使ってシートを作ると，ディスク上には次のような名前のファイルが作成されます．

MEMO.IDX
MEMO1.MEM　MEMO2.MEM　MEMO3.MEM……MEMO60.MEM

これらはすべて，アスキー形式のファイルで，システムメニューのTYPE，PRINT，BASICのTYPE, PRINT#, INPUT#等で扱えるようになっています．

MEMO1.MEM，MEMO2.MEM等にはシートの内容が入っていて，このなかにはシートのそれぞれのセル（マニュアルではウィンドウとなっていますが，ここでは一般的なセルという用語を使うことにします）の内容が入っているものと，シートのフォーマットが入っているものとがあります．そしてMEMO.IDXは名前が示すとおりインデックスファイルで，MEMO××.MEMというファイルに対する目次のようなものです．

MEMO.IDX

具体的にはMEMO.IDXは，

〈ファイル名〉，〈ファイル番号〉[，〈モード〉] CR／LF

といったものがいくつか集まった形になっていて，MEMOで作られる1つのシートに対して上のようなものが，2つ書き込まれています．1つは〈ファイル名〉が，そのシートの名前（シートの名前とはMEMOのインデックスモードのとき入力したファイル名のことを指します）になっているもの．2つ目は〈ファイル名〉が，シートの名前に，

.FORM

を付けたものになっているものです．たとえばSEISEKI 2という名前で，シートが作られているとすると，MEMO.IDXのなかには，

SEISEKI2, ××××
SEISEKI 2.FORM, ××××

といったものが書き込まれていることになります．〈ファイル名〉がシートの名前そのものになっているものは，MEMO××.MEMのうちセルの内容が入っているものに対するインデックス．〈ファイル名〉に.FORMの付いたものは，MEMO××.MEMのうちシートのフォーマットが入っているものに対するインデックスです．

〈ファイル番号〉はMEMO××. MEMの××を示します．たとえば，

```
SEISEKI2.FORM, 7 ,128
SEISEKI2, 8, 3
```

といったものが書き込まれていたとすると，MEMO 7. MEM は SEISEKI2 というシートのフォーマットが入っているファイル，MEMO8. MEM は SEISEKI2 のセルの内容が入っているファイルということになります. MEMO がシートの内容をセーブするときは，そのとき使われていない〈ファイル番号〉のうち一番小さいものをフォーマットのファイルに，その次に小さい〈ファイル番号〉をセルの内容が入っているファイルに割り当てますが，自分で MEMO. IDX を操作するときは，こうしなくてもいっこうに差支えありません.

最後の〈モード〉には，

bit 7 〈ファイル名〉をインデックスモードのとき表示するかどうかを選択する
1だと表示されない
bit 6 未使用
bit 5 未使用
bit 5 未使用
bit 4 未使用
bit 3 未使用
bit 2 未使用
bit 1 bit 1とbit 0で計算のモードを示す
bit 0 00 Nocal モード 01 Manu モード 10 未使用 11 Auto モード

のような8ビットを10進数で表したものが入っています.〈ファイル名〉が××.FORM となっているものはbit 7が立っていて， .FORM が付いていないものは，bit 7は立っていません.

セルのファイル

シートのなかのそれぞれのセルの内容の入ったファイルは，

〈行番号〉,〈列番号〉,［〈カラーコード〉］，〈セルの内容〉 CR／LF

といったものが集まった形になっています.〈カラーコードは〉，

bit 7 未使用
bit 6 未使用
bit 5 1だと計算結果であることを示す．Auto, Manuモードのファイルを Searchした

ものをセーブすると，数値の入っているセルは皆このビットが立つ

bit 4 未使用

bit 3 未使用

bit 2
bit 1
bit 0
} bit 2，1，0でそのセルの文字の色を表す BASICなどのカラーコードの各ビットを反転させたもの

といった8 bitを10進数で表したもので，0のときは省略できます.

アスキーファイルになっている関係上，〈セルの内容〉にコントロールコードが含まれてはいけません．また〈セルの内容〉の前までにはスペースが入ってはいけません.

フォーマットのファイル

シートのフォーマットの入ったファイルは，

〈機能番号〉，〈列番号〉,，〈内容〉CR／LF

といったものが集まった形になっています.

機能番号は1から4までで，それぞれ次のような意味を持っています.

1：〈内容〉が〈列番号〉の列の見出し（画面上でそれぞれの列の一番上に出ているもの）であることを示す

2：〈列番号〉の列が右詰めであることを示し，〈内容〉は「L」になっている．この場合〈内容〉は本来冗長な情報であるが，ここが「L」になっていないと，無視されてしまう

3：〈内容〉が〈列番号〉の列のFormula のUsing の内容であることを示す

4：〈列番号〉の列がLimitの列であることを示し，〈内容〉は「E」になっている この「E」も2の場合と同様に冗長であるが，なくてはならない

なお，参考までにリスト5-10にMEMO.IDX の内容，リスト5-11にMEMO7. MEM，リスト5-12にMEMO8. MEM の内容をそれぞれ表示しておきます.

List 5-10

```
SEISEKI.FORM,5,128
SEISEKI,6
SEISEKI2.FORM,7,128
SEISEKI2,8,3
SEISEKI1.FORM,1,128
SEISEKI1,2
```

List 5-11

```
1,1,,ﾅﾏｴ
1,2,, ﾊﾞﾝｺﾞｳ
1,3,,   ｺｸｺﾞ
1,4,,     ｻﾝｽｳ
1,5,,       ﾘｶ
1,6,,     ｼｬｶｲ
1,7,,!
1,8,,   ｺﾞｳｹｲ
1,9,,     ﾍｲｷﾝ
4,9,,E
```

List 5-12

```
1,1,,ﾕｳｺ
1,2,,1
1,3,,100
1,4,,12
1,5,,11
1,6,,83
1,7,,!
2,1,,ｱｲｺ
2,2,,2
2,3,,80
2,4,,34
2,5,,22
2,6,,42
2,7,,!
3,1,,ﾘｮｳｺ
3,2,,3
3,3,,70
3,4,,56
3,5,,33
3,6,,47
3,7,,!
4,1,,ｹｲｺ
4,2,,4
4,3,,60
4,4,,78
4,5,,44
4,6,,74
4,7,,!
5,1,,ﾏﾁｺ
5,2,,5
5,3,,50
5,4,,90
5,5,,55
5,6,,58
5,7,,!
6,1,,ｶｽﾞﾐ
6,2,,6
6,3,,40
6,4,,15
6,5,,66
```

```
6,6,,67
6,7,,¦
7,1,,ﾋﾛｺ
7,2,,7
7,3,,30
7,4,,26
7,5,,77
7,6,,65
7,7,,¦
8,1,,ﾏﾕﾐ
8,2,,8
8,3,,20
8,4,,37
8,5,,83
8,6,,43
8,7,,¦
9,1,,'------------------------------------------------
9,7,,+
10,1,,ｺﾞｳｹｲ
10,3,,=SUM(C1:C8)
10,4,,=SUM(D1:D8)
10,5,,=SUM(E1:E8)
10,6,,=SUM(F1:F8)
10,7,,¦
10,8,,=SUM(C1:F8)
11,1,,ﾍｲｷﾝ
11,3,,=AVG(C1:C8)
11,4,,=AVG(D1:D8)
11,5,,=AVG(E1:E8)
11,6,,=AVG(F1:F8)
11,7,,¦
11,9,,=AVG(C1:F8)
14,9,,
```

(3) プログラム例

MEMOで作成したSEISEKI2のファイルからデータを読み出し，グラフ処理して表示するプログラムをリスト5－13に表示します．このプログラムは全部で5つのブロックからなり，各部の構成は以下のようなものです．

① MEMO.IDXのなかからMEMOで作成したファイル名を検出する
② MEMOで作成した該当ファイル名のファイルから，セルの内容を読み出す
③ MEMOで作成した該当ファイル名のファイルから，フォーマットのデータを読み出す
④ 上記②，③で得られたデータを集計し，ディスプレイ上に表示する
⑤ データをグラフ処理し，棒グラフの形でディスプレイ上に表示する

プログラムの構成はおおむねこのようなものですので，以下行番行順に，要点を絞って説明してゆきます．

80　　　　：MEMO.IDX のファイルをオープンします.

90～170　　：この9ステップで，MMD＄に"．"を含まない行の〈ファイル名〉が記憶されます.
これはリスト5－10の，

SEISEKI
SEISEKI 1
SEISEKI 2

の3つのファイル名がMMD＄(NUM) に記憶されることを示します.

180～220　：OPEN したファイルを CLOSE し，かつ MMD＄(NUM) に記憶したファイル名を出力します.

240～340　：キーボードから入力された番号のファイル名を，MEMO．IDX から探し出します. 発見されたファイル名は DNAME＄ に記憶され，リスト5－10で言うと，SEISEKI 2，8，3 が DNAME＄ に記憶されることになります.

410～460　：見つけられたファイル名から，MEMO ××.MEM に対応するファイル名(××には最初の"，"の後の数値――ここでは SEISEKI 2，8，3 のため，8になる――が入ります)を作成します.

480～640　：MEMO××.MEM のファイルから，解析に必要なデータを読み取ります. うち，560～570行は，['] のデータ終了マークの検出，580～590行は [¦] を検出し，個人別データの区切りマークを検出します. DL＄ にはリスト5－12の1行分の文字列が記憶されており，これは1180～1240行のサブルーチンで，解析に不必要な文字等は削除されることになりますし，解析に必要なデータは HITO＄(NIN，KOU) で記憶されます.

680　　　　：MEMO.IDX から検出したファイル名を，ディスプレイ上に表示します.

690～760　：MEMO.IDX からシートのフォーマットの入っているファイル名を探し，対応するディレクトリ中のファイル名（ここでは MEMO 7 .MEM）を作成して OPEN し，MSG＄(INUM)に必要な項目を記憶します.

770　　　　：700で OPEN したファイル名を CLOSE します.

ここまでが，MEMO の内容を読み取るためのプログラムで，これ以降は，

780～1130　：生徒の成績を集計し，ディスプレイ上に表示します.

1260～1680：成績の棒グラフを，ディスプレイ上に表示します.

以上が MEMO と BASIC のリンクプログラムの概要です. すでにお気付きと思いますが，行番号780以降のプログラムは，データの持つ質や内容に応じて適宜作り直すべきもので，その方が視覚効果を高める上で，より有効であると思われます.

このように，それ自体なかなか結構な機能を備えた MEMO に，BASIC によるプログラムをり

ンクさせることで，さらに強力な簡易言語に仕上げることができるわけです．ここでは力点を視覚効果の方に置きましたが，ここで試みたデータの読み出しをさらに強化することで，たとえば高度な数値管理であるとかシミュレーションなどにも，MEMOの応用範囲を拡大できるものと思います．

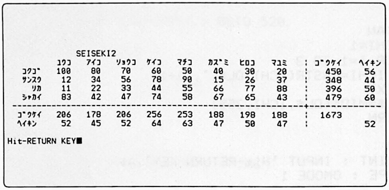
List 5-13

```
 10 '
 20 ' --- MEMEO = BASIC ---
 30 '
 40 *MAIN
 50  WIPE : CLEAR : DIM MSG$(20),MMD$(30)
 60  CONSOLE 40
 70  PRINT : PRINT 'FILE MENU' : PRINT
 80  OPEN /I #1,'MEMO.IDX'
 90  NUM=1
100  WHILE NOT (EOF(1))
110    INPUT /L #1,MENU$
120    A1=INSTR(MENU$,',')-1
130    B1=INSTR(MENU$,'.')
140    IF B1<>0 THEN 100
150    MMD$(NUM)=LEFT$(MENU$,A1)
160    NUM=NUM+1
170  WEND
180  CLOSE
190  PRINT ' No. File name'
200  FOR I=1 TO NUM-1
210    PRINT TAB(3);I;' ';MMD$(I)
220  NEXT
230 '
240 *FIND
250  PRINT : INPUT 'FILE No. ';MNON
260  IF MNON<=0 THEN PRINT : PRINT 'PROGRAM END ' : END
270  IF MNON>NUM THEN *NOFIL
280  OPEN /I #1,'MEMO.IDX'
290  MNAME$=MMD$(MNON)
300  WHILE NOT (EOF(1))
310    INPUT /L #1,DNAME$
320    LNO=INSTR(DNAME$,MNAME$+',')
330    IF LNO<>0 THEN *FIND2
340  WEND
350 '
360 *NOFIL
370  BEEP : CCOLOR 4
380  PRINT : PRINT 'File not found':CLOSE
390  CCOLOR 7 : PRINT : GOTO 70
400 '
410 *FIND2
420  CCOLOR 3 : PRINT : PRINT  'Found:';MMD$(MNON) : CCOLOR 7
430  LNO1=INSTR(DNAME$,',')+1 : DN$=MID$(DNAME$,LNO1,2)
440  IF RIGHT$(DN$,1)=',' THEN DN$=LEFT$(DN$,1)
450  MA$='MEMO'+DN$+'.MEM' : DN=VAL(DN$)
460  CLOSE
470 '
480 *YOMU
490  OPEN /I #1,MA$
```

```
 500  NIN=0
 510  CHK=0
 520  WHILE NOT (EOF(1))
 530    NIN=NIN+1
 540    FOR KOU=1 TO 128
 550      INPUT /L #1,DL$
 560      EN1=INSTR(DL$,'´')
 570      IF EN1<>0 THEN CLOSE : GOTO *EFILE
 580      PAS=INSTR(DL$,'¦')
 590      IF PAS<>0 THEN CHK=CHK+1 : GOTO 520
 600      GOSUB *HENKAN
 610      HITO$(NIN,KOU)=DL$
 620    NEXT
 630  WEND
 640  CLOSE
 650 '
 660 *EFILE
 670  WIPE : CONSOLE 80
 680  PRINT  MMD$(MNON)
 690  MNAME1$=LEFT$(MA$,4)+STR$(DN-1)+'.MEM'
 700  OPEN /I #1,MNAME1$
 710  WHILE NOT (EOF(1))
 720    INUM=INUM+1
 730    INPUT /L #1,DL$
 740    GOSUB *HENKAN
 750    MSG$(INUM)=DL$
 760  WEND
 770  CLOSE
 780  PRINT  '              ';
 790  FOR K=1 TO CHK
 800    PRINT USING '¦      ¦',HITO$(K,1);
 810  NEXT
 820  PRINT '¦ ｺﾞｳｹｲ    ﾍｲｷﾝ'
 830  FOR I=3 TO 6
 840    PRINT  USING '¦       ¦',MSG$(I);
 850    GO1=0
 860    FOR J=1 TO CHK
 870      PRINT USING '######',VAL(HITO$(J,I));
 880      GO1=GO1+VAL(HITO$(J,I))
 890    NEXT
 900    GO2=GO1/CHK
 910    PRINT '   ¦'; : PRINT USING '######',GO1,GO2
 920  NEXT
 930  PRINT ' ';STRING$(57,'-');'+';STRING$(13,'-')
 940  PRINT '  ｺﾞｳｹｲ';
 950  GO4=0
 960  FOR I=1 TO CHK
 970    GO3=0
 980    FOR J=3 TO 6
 990      GO3=GO3+VAL(HITO$(I,J))
1000    NEXT
1010    PRINT USING '######',GO3;
1020    MEO(I)=GO3
1030    GO4=GO4+GO3
1040  NEXT
1050  PRINT '   ¦'; : PRINT USING '######',GO4
1060  PRINT '  ﾍｲｷﾝ ';
1070  HEI=0 : HEI1=0 : HEI2=0
```

```
1080  FOR I=1 TO CHK
1090    HEI=MEO(I)/4 : HEI1=HEI1+HEI
1100    PRINT USING "######",HEI;
1110  NEXT
1120  HEI2=HEI1/CHK
1130  PRINT "  ¦       "; : PRINT USING "######",HEI2
1140  GOSUB *HYOU
1150  GOTO *MAIN
1160  END
1170 '
1180 *HENKAN
1190    ICHI=1
1200    FOR I=1 TO 3
1210      ICHI=INSTR(ICHI,DL$,",")+1
1220    NEXT
1230    DL$=MID$(DL$,ICHI,255)
1240  RETURN
1250 '
1260 *HYOU
1270    PRINT : INPUT "Hit-RETURN KEY",A$
1280    WIPE : GMODE 1
1290    DEF PALETTE 1,3,4,7
1300    GCOLOR 1,3,3,0
1310    GCLEAR 1,0
1320    BOXE(0,0)-(319,199),4
1330    LINE(0,20)-(319,20),5
1340    LINE(50,20)-(50,199),5
1350    LINE(301,20)-(301,199),5
1360    LINE(176,20)-(176,199),5
1370    FOR CFT=0 TO 3
1380      BOXF (60*CFT+80,8)-(60*CFT+100,13),CFT+2
1390      BOXE (60*CFT+80,8)-(60*CFT+100,13),0
1400    NEXT
1410    IYSD=0
1420    FOR YSFT=1 TO CHK
1430      IYSD1=IYSD
1440      IYSD=IYSD1+4
1450      CO=2
1460      FOR XSFT=3 TO 6
1470        XBAR=VAL(HITO$(YSFT,XSFT))
1480        IYSD=IYSD+3
1490        IXB=XBAR*2.5+51
1500        IYB=IYSD+30
1510        BOXF(51,IYB)-(IXB,IYB+3),CO
1520        BOXE(51,IYB)-(IXB,IYB+3),0
1530        CO=CO+1
1540      NEXT
1550    NEXT
1560    PY=3
1570    FOR HI=1 TO CHK
1580      PY=PY+2
1590      PRINT (5,PY) USING "¦     ¦",HITO$(HI,1)
1600    NEXT
1610    IXX=0
1620    FOR KI=3 TO 6
1630      IXX=IXX+15
1640      PRINT (IXX-4,1) USING "¦      ¦",MSG$(KI)
1650    NEXT
```

```
1660     LOCATE (15,23)
1670     INPUT "Hit-RETURN KEY",A$
1680   RETURN
```

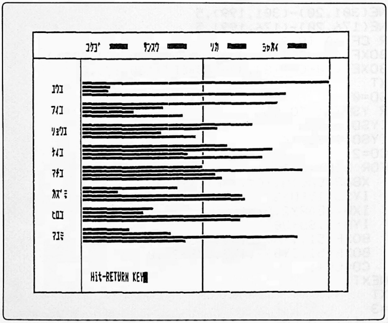


SEISEKI2

	ﾕｳｺ	ｱｲｺ	ﾘｮｳｺ	ｹｲｺ	ﾏﾁｺ	ｶｽﾞﾐ	ﾋﾛｺ	ﾏﾕﾐ	ｺﾞｳｹｲ	ﾍｲｷﾝ
ｺｸｺﾞ	100	80	70	60	50	40	30	20	450	56
ｻﾝｽｳ	12	34	56	78	90	15	26	37	348	44
ﾘｶ	11	22	33	44	55	66	77	88	396	50
ｼｬｶｲ	83	42	47	74	58	67	65	43	479	60
ｺﾞｳｹｲ	206	178	206	256	253	188	198	188	1673	
ﾍｲｷﾝ	52	45	52	64	63	47	50	47		52

Hit-RETURN KEY■

まず成績を表の形で表示する



成績をグラフにする

6章　グラフィックスの世界

SMC-777の特徴のひとつに，グラフィックス機能の充実ぶりを挙げないわけにはいかないでしょう．オプションとはいえ比較的低価格でサポートされるカラーパレットは，4096色から16色を選べるという機能を持ち，SMC-777のグラフィックス機能を，この価格帯のマシンとしては傑出したものとしています．また，777 BASICのグラフィックス関係の命令も充実しており，SMC-777のグラフィック機能を十分に引き出すことができます．

本章では，まずコンピュータ・グラフィックス全般の動向を見渡した後，グラフィックスの基礎，777 BASICのグラフィック関連コマンドの紹介，カラーパレット・テクニック，グラフィックス上級編へと進み，さらにソニーから発売されているSMCグラフィックスエディター（SMW-S001D）の紹介と，これで作成したデータをBASICプログラムで有効利用した楽しいゲームと，N_{88}-BASICからの移植テクニックの紹介を行います．

6-1　コンピュータ・グラフィックスの役割り

人間とは不思議な生きもので，身に備わった五感のうちでも，とりわけ視覚に依存する傾向があるようです．見ること，それは人間にとって，ものごとを認識する上で最もてっとり早い手段です．しかも，視覚によって認識されたものに対し，人間はさらにイメージを膨らませ，思考を始める手懸りともします．つまり見ることとは，人間が認知行動を始めるための，最初のとっかかりというわけです．

人間とコンピュータの関係にも，この視覚が重要な役割りを果たします．なにしろマンマシン・インターフェイスにおける機械からのメッセージの出力が，ディスプレイなのですから当然のことではあります（まァ，音での出力もあるにはあるが…）．人間が文字を整合性のある記号として認識し，また描かれた絵に対してもメッセージを読み取る能力がある以上，画像出力は最も強力な情報伝達手段であるわけです．

非常に高速で計算し，かつ大量のデータを蓄積することのできるコンピュータにとって，人間の認知パターンから考えると，視覚に訴えることが最も効果的なメッセージ伝達手段になるわけ

で，視覚に対してどのように情報を加工してやるかが，重要なポイントとなってきます．

コンピュータ・グラフィックスは，このようなデータ・プレゼンテーションに際して強力な武器となります．たとえば最近のOAで顕著なように，数値データを視覚的に処理したグラフで見せてやるような場合，メッセージの受け手は，単に数字を羅列されたときに比べて，はるかに効率的にデータを認知します．ここでひとつ気が付くのは，同じ数値データが基本になっていようとも，グラフィック処理の手際のよし悪しにより，受け手の認識は大きく変化するということです．だからコンピュータを使ってデータ処理（つまり，何かをすることですが）をしてゆく場合，効果的なグラフィックスというのは必要不可欠の技術であると考えてよいのではないでしょうか．

一方では，すでにCGが機械を使ったお絵描きの試みの域に留まらず，立派な芸術の域に近付いていることも，忘れてはならないでしょう．過去のあらゆる表現媒体と比較しても，コンピュータを用いてのグラフィックスはその表現能力において，遜色ないどころか，それらを凌駕さえするかもしれない潜在力を秘めています．絵画にしろ映像にしろ，つきつめてみれば似たようなものであり，原点的発想を持ってすれば，CGが既存の表現媒体と比肩し得るのは自明の理．むしろその使い勝手のよさから考えれば，より広範な人々に表現のチャンスを与えていると言えるでしょう．

まァ，こういったコンピュータ・グラフィックスの役割りを抜きにしても，パソコンを使ってグラフィックスするのは，とても楽しいものです（みなさん御存知でしょうが）．しかもSMC-777は映像分野に強いソニーの製品らしく，グラフィックスに関して非常に豊富な機能を持っています．標準のモニタ，KX-13CD 1 の発色のよさともあいまって（なにしろ表現力に富むアナログRGBモニタなので…），777のグラフィックスには，他機種とひと味違ったおもしろさがあるようです．なにはともあれ，まずは777グラフィックスをお試しください．

次の，「グラフィックスの基礎」の節は，もし理屈は好きじゃないというのなら飛ばしてプログラミングに入っていただいても結構です．まず，プログラムを打ち込んでグラフィックスが画面に現れるのを見て，しかるのち興味が湧いてきたら少し勉強してみてください．

6-2　グラフィックスの基礎

パソコンでグラフィックスを行う場合，基礎知識として，

① 画面構成
② キャラクタ表示
③ グラフィック表示
④ グラフィック座標系
⑤ 三角関数

この５本の概念が，グラフィックスを実践する上での基礎的な柱となります．この５本の基礎知識はいずれも結構難しいため，実際のプログラミングを通して覚えてゆくことが望ましいと考えます．ですからこの節における説明ではごく一般的なレベルのものをマニュアル的に解説し，読者諸兄が技術の習熟の進展に合わせ，適宜該当項目に戻ってこられるように構成します．

(１) 画面構成

SMC-777においては３つの画面を重ね合わせて，ひとつの完成された画面を作り出しています．これらの画面は，ボーダーエリア，グラフィックス表示面，キャラクタ表示面と呼ばれています．

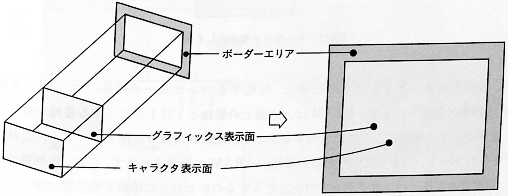


図6-1　３つの画面

画面の周辺部分を構成しているボーダーエリアは，文字や図形を表示することはできませんが，色を指定することが可能なので，画面全体の彩色構成に重要な役割りを果たし，グラフィックス完成時の画面のよし悪しを左右する重要な部分です．

また，キャラクタ表示面(スクリーン)は文字や記号を，グラフィックス表示面(スクリーン)は線などを表示する文字どおりの役割りを果たします．この2つの表示面は，画面上ではいかにも同一面上に表示されているように見えますが，コンピュータ内部では，別々の情報として扱われています．

この２つの表示面ではキャラクタの表示面に優先権があり，文字や記号の上に線が重なるということはありません．

(２) キャラクタ表示

画面上に，キャラクタ（文字，記号）を表示する役目を負っているのは，キャラクタRAM (CRAM)，アトリビュートRAM（ARAM)，プログラマブル・キャラクタジェネレータ（PCG）と呼ばれる３つの部分です．

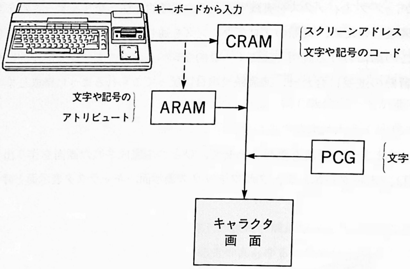
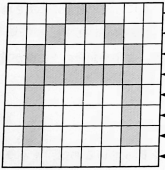


図6-2 キャラクタ表示のしくみ

まず，表示させせようとする文字や記号は，対応するコンピュータ内部のコードに変換されCRAMの内部に記憶されます．CRAMは，画面上の番地と1対1で対応する番地を持っているので，文字コードと同時に，どこに表示するかという場所の情報も記憶しています．これと同時に，アトリビュート，つまり文字や記号の属性がARAMに記憶されます．これは，何色で表示するか，点滅させるか，バックグランドの色はどうするのかといった情報を取り扱います．キーボードから，これらの文字情報が入力された場合，CRAM，ARAMに取り込まれ，画面に表示されるわけですが，文字や記号のコードは，そのままでは画面に表示できません．このコードに対応する文字や記号が，実際にどういう形をしているのかを記憶しているのがPCGです．CRAMからコードを受け取ったPCGは，コードに対応した文字や形の情報にARAMのアトリビュート情報を付加して画面上に送り出します．

777 BASICでは，キャラクタ表示面の表示文字数をソフト上の操作で自由に変更できます．



第1バイト	0	0	0	1	1	0	0	0
第2バイト	0	0	1	0	0	1	0	0
第3バイト	0	1	0	0	0	0	1	0
第4バイト	0	1	1	1	1	1	1	0
第5バイト	0	1	0	0	0	0	1	0
第6バイト	0	1	0	0	0	0	1	0
第7バイト	0	1	0	0	0	0	1	0
第8バイト	0	0	0	0	0	0	0	0

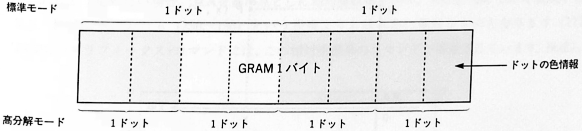
キャラクタコード65の図形情報

図6-3 PCGの図形情報と画面上の表示

CONSOLE というコマンドを使うことで，80字×25行，40字×25行のふたつのモードを使うことが可能なのです．

（3）グラフィック表示

SMC-777のグラフィックスは，グラフィック RAM（GRAM）が一手に引き受けています．GRAM は，画面上の１ドットずつの位置や色の情報を持ち，対応するドットを塗りつぶしてゆきます．この場合の表示ドットには，分解能に応じて２つのモード ── 標準モードと高分解能モード ── が用意されています．標準モードが縦200×横320ドット（16色）であるのに対し，高分解能モードの方は，縦200×横640ドット（４色）により構成されています．



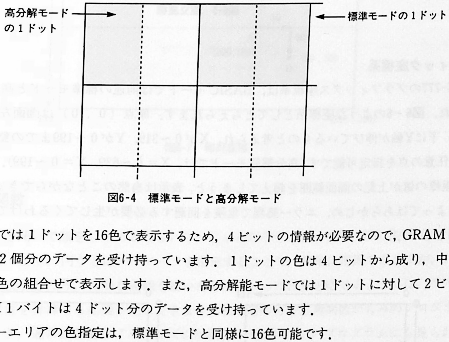


図6-4　標準モードと高分解モード

標準モードでは１ドットを16色で表示するため，４ビットの情報が必要なので，GRAM の１バイトはドット２個分のデータを受け持っています．１ドットの色は４ビットから成り，中間色，赤，緑，青の色の組合せで表示します．また，高分解能モードでは１ドットに対して２ビット，つまり GRAM １バイトは４ドット分のデータを受け持っています．

なおボーダーエリアの色指定は，標準モードと同様に16色可能です．

（4）グラフィック座標系

グラフィックス画面における図形の表示は，画面を構成するドット（画素＝pixcel（ピクセル））に指定の色を与えることで行われます．このピクセルは縦方向と横方向に整然と配列することにより任意の位置に配することができ，この縦横方向の位置決定に座標の概念を用います．

座標系

空間もしくは平面上の任意の位置を表す数字の組を，座標と呼んでいます．図6－5に示すように，原点Oで直角に交わる座標軸を持ち，点Pを(x，y)で表現できるような座標系を直交座標(デカルト座標）と言います．直交座標では，点PからX軸に下した垂線とX軸の交点の値をX座標，Y軸に下した垂線とY軸の交点の値をY座標としています．座標系にはこれ以外に，3次元の直交座標や極座標系なども用いられます．

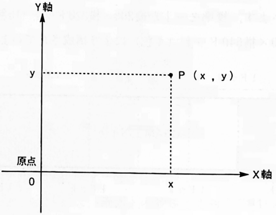


図6-5　直交座標

グラフィック座標系

SMC-777のグラフィックス座標系は，BASICモードでは前述の標準モードと高分解能モードに分かれ，図6－6のような座標系としてとらえられます．原点（0，0）は画面左上にあり，右にX軸，下にY軸が伸びているものと考えられ，Xが0～319，Yが0～199までの整数の組で，画面上の任意の点を指定可能です(高分解能モードでは，X＝0～639，Y＝0～199)．このときX座標，Y座標の値が上記の画面範囲を超えてしまうと，表示は当然のことながらできません．プログラムによってはあらかじめ，エラー処理で危険を回避する必要が生じてくるわけです（後に詳しく説明します）．

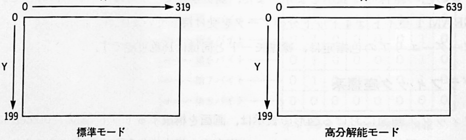


図6-6　グラフィック座標系

絶対座標と相対座標

原点(0, 0)に対してX座標値, Y座標値で任意の位置を表す直交座標のような座標系では, 原点を基準として見たときの任意の座標を絶対座標と呼んでいます. これに対し, 基準を他のものに置き換えて表現するような座標系を相対座標と呼び, グラフィックスにおいては何かと重宝する考え方となります.

絶対座標はたとえて言えば, 自分の住所を何町の何丁目何番地と, あらかじめ決められている絶対値で表現するようなもの. これに対して相対座標は, オレの家は角の煙草屋から5軒目だ, というように, 任意の基準を定めた上で表現する方法です. 図6-7で示すように, 点Bが常に点A(200, 100)の位置からX軸方向にプラス50ドット分, Y軸方向にプラス50ドット分移動した場所にあるよう定めたい場合, 点Aを基準点とした相対座標系を作り, 点Bを(50, 50)で表現する方が, 絶対座標における(250, 150)という表現よりもはるかに便利なものとなります(777 BASICのグラフィックス・コマンドには, この相対座標系のコマンドが用意されています. 後述).

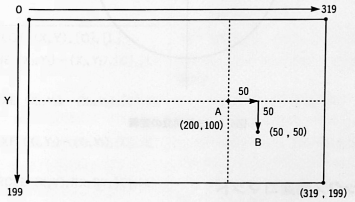


図6-7 相対座標

(5) 三角関数

三角関数, と聞いて顔をしかめる方もいらっしゃるだろうと思います. 777 BASICでは, 三角関数を知らなくとも, 円などはパラメータを指定することで, 簡単に描画できますが, 円周上にキャラクタを表示したり, 特殊な曲線を描いたり, といった仕事をさせるときには, サイン, コサイン等で位置を計算する必要が生じてきます. 初心者の方が三角関数がらみのプログラムを作ろうとするとき, 通常定型的なアルゴリズムが数多くのパソコンのプログラムにも見られますので, あちこちのプログラム例から見つけ出して, 習作し, 自分なりに変更したり修正したりして身に付けてゆくことが早道でしょう. では, 以下に簡単な三角関数のあらましを説明しましょう.

三角関数の定義は図6-8に示すように, 円周上の点Pと円の中心Oとを結んだ線分と, 点PからX軸に下した垂線の交点と原点Oを結んだ線分とによって成立する三角形において, $\sin\theta = PA/OP$, $\cos\theta = OA/OP$, $\tan\theta = PA/OA$となります. 円の半径を1として考えた場合, $\sin\theta$, $\cos\theta$の

値は，それぞれPA，OAの長さとなります．またθは角度を表しますから，度あるいは弧度として扱わねばなりませんが，コンピュータで角度を扱うときは，一般的には弧度法を用います．弧度法の単位ラジアンはπを使った角度の表現法で，度を弧度に変換するには，$\pi = 180°$により，弧度＝度×π/180で得られます．三角関数はCGにとって重要な意味を持つため，必要に応じて数学関係の書籍の参照が望まれます．

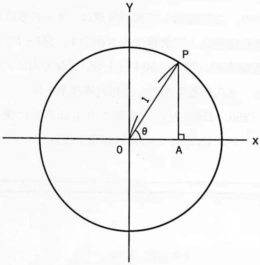


図6-8　三角関数の定義

6-3　グラフィックス関連コマンド

いよいよ SMC-777 で実際にグラフィックスを行うことにします．この節では，グラフィックスに用いる各種のコマンドの機能を解説し，それに見合った短いプログラムを試してみることで，コマンドに対する理解を深めるとともに，グラフィックス技術の向上を目指します．最初に，グラフィックスの基本のそのまた基本である，画面クリア，条件設定，表示等のコマンドの大筋を説明しておきましょう(表6－1)．

画面クリアはビデオRAMの内容をクリアする命令で，新たに図形を描くとき，以前に描いたものを消すために用います．命令には3種類あって，テキストをクリアするCCLEAR，グラフィックスをクリアするGCLEAR，そしてグラフィックスとキャラクタを共にクリアし，かつその他の設定も初期化するWIPEです．

条件設定は図形を描く前に必要に応じて行うもので，たとえば高分解能モードで作画したいとき，GMODEで分解能を変えるような使い方があります(標準モードに戻す場合もGMODEで指定)．このほかにも，座標の原点を変えるGLOCATE，表示命令時のグラフィック表示色を変えるGCOLOR(ボーダーエリアの色の変更も含む)等が挙げられます．

分 類	書 式	機 能
画面クリア	WIPE CCLEAR [B],[E] GCLEAR [C],[L]	テキスト画面，グラフィックス画面をクリアする テキスト画面をクリアする B：消去開始行，E：消去行数 C,Lで決定される色に画面をクリアする C：カラーコード，L：論理演算コード
条件設定	GMODE M,[T] GLOCATE (X,Y),[Xu],[Yu] GCOLOR [F],[BG],[B],[L]	分解能の設定をする M：M=1 標準モード，M=2 高分解能モード T：表示カラータイプ(M=2の時のみ指定) 座標原点を (X,Y) に移動する Xu：X軸の方向を指定，Yu：Y軸の方向を指定 表示カラーの指定のデフォルト値を与える F ：表示カラーコード BG：バックグラウンドカラーコード B ：ボーダーエリアカラーコード L ：論理演算コード
表 示	GPLOT (X,Y),[C],[L] LINE $(X_1,Y_1)-(X_2,Y_2)$,[C],[L] BOXE $(X_1,Y_1)-(X_2,Y_2)$,[C],[L] BOXF $(X_1,Y_1)-(X_2,Y_2)$,[C],[L] CIRCLE (X,Y),RX,RY,[C],[L] PAINT (X,Y),$[C_1]$,$[C_2]$	(X,Y)座標にC, Lで決定される色の点を打つ (X_1,Y_1)，(X_2,Y_2)の間をC, Lで決定される点で直線補間する C, Lで決定される色の箱を描く (X_1,Y_1)，(X_2,Y_2)は描こうとする箱の対角の座標 C, Lで決定される色の箱を描きなかを塗りつぶす (X_1,Y_1)，(X_2,Y_2)はBOXEのときと同じ意味 C, Lで決定される色の円を描く (X,Y)：円の中心座標 RX：X方向の半径，RY：Y方向の半径 (RX,RYのうちどちらか一方は省略できる) 指定した座標(X,Y)を含み，C_2で示される色を境界色として，そのなかをC_1で示される色に塗りつぶす
そ の 他	GGET $(X_1,Y_1)-(X_2,Y_2)$,A%(0) GPUT $(X_1,Y_1)-(X_2,Y_2)$,A%(0), [L],[T] DRAW <サブコマンド列>	(X_1,Y_1)，(X_2,Y_2)を対角とする範囲の画面のデータを配列A%に取り込む GPUTで取り込まれたデータを(X_1,Y_1)，(X_2,Y_2)の範囲に表示する T：透明色コード サブコマンド列の指示に従い，図形を描く
関 数	GPOINT (X,Y) GPOSX GPOSY	グラフィック表示面の (X,Y) 座標のカラーコードを返す グラフィックコマンドで指定した最終指定座標又は，DRAWの現地点の座標のX座標を返す GPOSXと同様にY座標の値を返す

[ ]内は省略可

表6-1 グラフィックス・コマンド

表示は，実際に座標を指定して点や線を表示するための命令です．このなかには，点を打つためのGPLOT，線を引くLINE，箱を線描するBOXE，箱を描きかつ内部を塗りつぶすBOXF，円を描くCIRCLE，境界色の範囲内を塗りつぶすPAINTがあり，これらを組み合わせて使うことにより，自由な描画が可能となるわけです．

これらのほかに，画面上のドットのカラーコードをそのまま配列変数に読み書きするGGET，GPUTや，グラフィックス関係の関数としてGPOINT，GPOSX，GPOSYなどがあります．さらにSMC-777にはDRAWと呼ばれるタートル・グラフィックス機能を持つコマンドがあり，豊富なサブコマンド群により，多彩なグラフィックス処理を可能としています．では，次より代表的なコマンドを用い，実際にプログラムを作ってゆきましょう．

(1) GPLOT

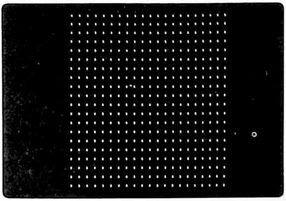
点を打つ命令であるGPLOTは，表示コマンド中，最もシンプル(と言うより原始的かも)な命令であり，他のすべての表示コマンドは，このGPLOTを用いて代替することが可能であると言えるほどです(そんなアホな真似をする人が居るとは思わんけど――)．GPLOTは4つのパラメータによって，任意の座標(X，Y)にC(カラーコード)，L(論理演算コード)で決定される点を打ちます．このC，Lは省略することができ，その場合直前に実行されたGCOLORもしくは初期設定に従います．論理演算コードの働きは，1がすでに表示されているカラーコードのNOT，2がすでに表示されている色と指定した色とのAND，同様に3がOR，4がXORとなっています．ですから画面上の(10，10)が赤く表示されているときに，GPLOT (10，10)，1，3とした場合，画面上の(10，10)には赤のコード4(0100)と青のコード1(0001)のOR式がとられ，赤紫(コード5(0101))が表示されます．このC，Lの意味は，ほかのコマンドにおいても同じになります．

では，以下に挙げるリスト6-1を，実際に入力してみてください．PLOTと名付けたように，このプログラムは，順番に点を打ってゆくだけの簡単なものです．

40行，50行にある二重のFOR～NEXTループ中のI，Jという変数が，X座標とY座標の値を決定します．各変数は，0から200までの値をとりますが，STEP 10つまり，10ドットずつにXとY

List 6-1

```
10 '-- PLOT --
20  DEF INT A-Z
30  WIPE : CURSOR OFF
40  FOR I=0 TO 200 STEP 10
50    FOR J=0 TO 200 STEP 10
60      GPLOT (J+60,I),2
70      GPLOT (I+60,J),10
80    NEXT J
90  NEXT I
100 GOTO 40
```



リスト6-1の実行画面

の値が与えられてゆきます．そして，60行，70行のGPLOTが画面上に点を打ちますが，座標値のI，Jがそれぞれ逆になっているので，赤い点は横，つまりX方向に，青い点は縦，つまりY方向にと打たれ，次々と重なって前に打たれた点にとって替っていきます．

このように，GPLOTは，任意の位置に任意の色の点を打つことができるので，使い方次第では色々な芸当をさせることができます．

(2) LINE

LINE文による命令は読んで字の如く直線を受け持っており，グラフィックス処理のなかで最も頻繁に用いるコマンドのひとつでしょう．LINE命令は，直線の始点と終点の座標を与える数値，(x_1, y_1)，(x_2, y_2) と，直線の色を決定する2つの数値，C，Lによって任意の2点間に指定色で直線を引きます．このとき (x_1, y_1) は省略することができその場合は前のグラフィックス・コマンドで指定された最終値か，DRAW の現地点が与えられます．試しにLINE文を使った多角形プログラムを実行してみましょう．リスト6-2を実行してください．大きさと頂点の数を与えることで，三角形であろうが何であろうが，任意の多角形が得られます．また頂点の数が20を超えるあたりから，多角形は限りなく円に近付いてゆきます．つまりLINE文は応用次第で，円を描くことにも利用できるということです．

ここで注意すべき点は，100行のLINE－(X1, Y1)という命令です．LINEを引くための始点となる座標が省略されていますね．LINE文では，始点座標が省略されると，直前に実行されたGPLOTの座標やLINE文の終点座標が新たなLINE文の始点座標として自動的に補完されるので，このプログラムのように線を連続して描く場合は便利です．

リスト6-3には，LINE文を使って円形に線を描くプログラムを紹介します．ごく単純なプログラムですが，全体の構成は，二重のFOR～NEXTループに収められ外側のFOR～NEXTで，21個の円を描くカウントをとり，ランダムに表示位置の座標を決定します．777 BASICのランダム文には，IRNDとRNDの2種類があり，ここではIRNDが使われます．通常，RNDは1以下の乱数を返しますが，IRNDは指定された引数以下の整数の乱数を返してきますので，プログラ

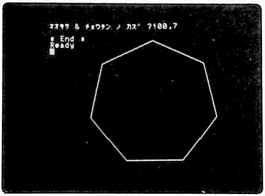
ム作成上なにかと便利です．60行の FOR ～ NEXTを見てもわかるとおり，STEP 10により，10度間隔で36本の線で円を構成し，SIN，COSの三角関数でLINE文の終点座標となる円周上の座標を計算します．円周上の座標を求める三角関数による式は，かなり定型化したアルゴリズムなので覚えておくと何かと役に立つでしょう．

List 6-2

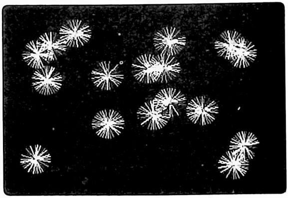
```
 10 '-- ﾀｶｸｹｲ --
 20  WIPE
 30  GLOCATE (160,100),1,-1
 40  INPUT "ｵｵｷｻ & ﾁｮｳﾃﾝ ﾉ ｶｽﾞ ";A,N
 50  X=0 : Y=A*.78
 60  GPLOT (X,Y),0,3
 70  FOR I=0 TO 360 STEP 360/N
 80    Y1=(A-J)*COS(I*PI/180)*.78
 90    X1=(A-J)*SIN(I*PI/180)
100    LINE -(X1,Y1)
110    X=X1 : Y=Y1
120  NEXT
```

List 6-3

```
 10 '-- line demo --
 20  WIPE
 30  RANDOMIZE
 40  FOR J=0 TO 20
 50    X=20+IRND(280):Y=20+IRND(160)
 60    FOR I=0 TO 360 STEP 10
 70      TH=I*PI/180
 80      X1=X+20*COS(TH)
 90      Y1=Y+20*SIN(TH)*.8
100      LINE(X,Y)-(X1,Y1),IRND(14)+1
110    NEXT I
120  NEXT J
```



LINEで多角形を描く



線を使って円を描く

（3）BOXE, BOXF

箱を描く命令には箱のなかが空で線のみのBOXE（Box Empty）と，箱の中を塗りつぶすBOXF（Box Full）の2種類があります．使い方はどちらも同じで，描こうとする箱の対角の座標をそれぞれ（x_1，y_1）（x_2，y_2）として定め，表示色を決めるC，Lを与えます．なお，このときの（x_1，y_1）は省略可能で，LINE命令で行われるのと同等の値が与えられます．

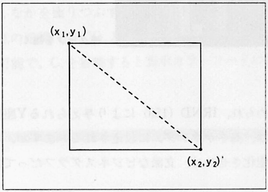


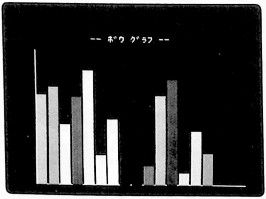
図6-9　箱を描くときに指定する点

さて，リスト6-4に紹介する棒グラフのプログラムはBOXFを用いた実に簡単なもので，グラフを描き易いようにGLOCATEで座標を変換して画面左下を原点(0，0)に設定しています．このGLOCATEは，原点を定める座標（X，Y）に，軸方向を定めるXu，Yuを与えることで使えます．軸方向はXuとYuに，それぞれ0を含む整数(正)を与えたときは右方向と下方向，整数(負)を与えたときに左方向と上方向になります．ここでは原点を画面左下（0，199）に定め，Y軸方向を上方向に指定してプログラムを書いています．

このプログラムでは，BOXFのX座標に相当する部分，つまりIとI+14によりひとつの棒グ

List 6-4

```
 10 ' -- ﾎﾞｳｸﾞﾗﾌ --
 20  WIPE
 30  CONSOLE 40 : CURSOR OFF
 40  GLOCATE (0,199),1,-1
 50  LINE (0,0)-(300,0)
 60  LINE (0,0)-(0,150)
 70  FOR I=1 TO 254 STEP 17
 80    CCODE = I/17+1
 90    LI = IRND(150)
100    BOXF(I,0)-(I+14,LI),CCODE
110  NEXT
120  LOCATE (10,4)
130  PRINT "-- ﾎﾞｳ ｸﾞﾗﾌ --"
140  EVAL INKEY$(1)
```



棒グラフを描く

ラフの幅（14ドット）が決められ，IRND（150）により与えられるY座標LIにより棒グラフの高さが与えられます．ここでは，高さがランダムに設定されていますが，これを与えられたデータやキーボードからの入力で変化させれば，立派なビジネスグラフだって簡単に書ける……ということにしておきましょう．

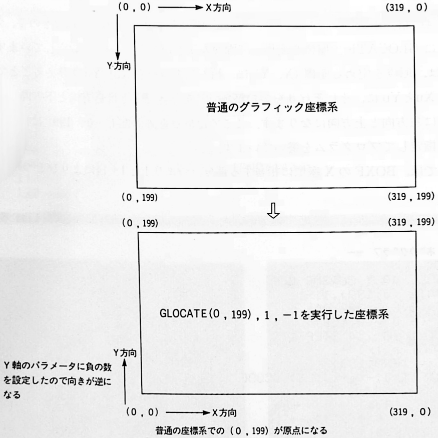


図6-10　GLOCATEによる座標系の変更

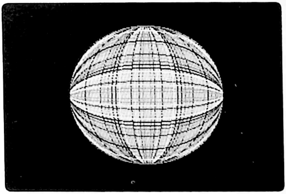
(4) CIRCLE

円を描くためのコマンド，CIRCLE は，円の中心座標 (X，Y) と，X軸方向の半径Rx，Y軸方向の半径 Ry に，円の色を定める C，L を与えることで，任意の位置に，指定色で指定の大きさの円を描く命令です．このとき，Rx もしくは Ry どちらか片方の省略ができ，その場合は指定軸の値を半径とした真円を描きます．

リスト6-5は，CIRCLE 文を用いてまり(昔なつかしいマリでして，ボールなどと呼んではいけませぬ)を描くプログラムで，なかを塗りつぶすために PAINT 命令を併用しました．この PAINT，塗りつぶしたい枠内の任意の座標 (X，Y) と塗る色C_1，枠の色 (境界色) C_2を与えることで機能します．C_1とC_2は省略が可能で，C_1を省略すると表示カラーコードに従い，C_2を省略するとC_1に従います．

List 6-5

```
 10 '-- MARI --
 20  WIPE
 30  GLOCATE (160,100)
 40  CIRCLE (0,0),100,,14
 50  PAINT (0,0),14,14
 60  FOR I=100 TO 1 STEP -5
 70    CIRCLE (0,0),I,100*.78,I/5
 80  NEXT
 90  FOR I=100 TO 1 STEP -5
100    CIRCLE (0,0),100,I*.78,I/5
110  NEXT
```



まりを描く

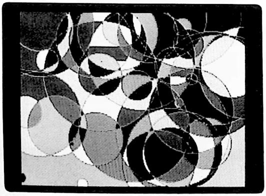
MARI プログラムでも，BOX で登場した GLOCATE が使われます．このプログラムでは，画面一杯に円を描くので原点を画面の中心に持ってきて使い易くしています．ふたつの FOR～NEXT ループは，それぞれ X 軸方向が縮む楕円と Y 軸方向が縮む楕円を担当しています．ここでは，楕円の短径を I の減少により表しているので，FOR～NEXT も数が小さくなっていくよ

うに設定しています．これを行うのがSTEP－5，つまり，Iの値が5ずつ減少してゆくわけです．

リスト6-6もまた，CIRCLEとPAINTを用いたプログラム例ですが，こちらは乱数で円の位置と大きさ及びPAINT色をランダムに決定させて描画してゆくものです．

List 6-6

```
 10 '-- CIRCLE PAINT --
 20 CURSOR OFF
 30 WIPE
 40 FOR I=1 TO 40
 50   X=IRND(319) : Y=IRND(199) : R=IRND(80) : CCODE=1
 60   CIRCLE (X,Y),R,,CCODE
 70 NEXT
 80 WHILE 1
 90   X=IRND(319) : Y=IRND(199) : CCODE=IRND(14)+2
100   PAINT (X,Y),CCODE,1
110 WEND
```

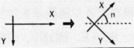
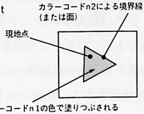
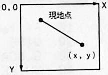
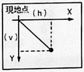
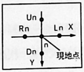
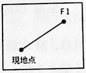
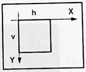
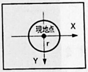


CIRCLEとPAINTによる模様

(5) DRAW

SMC-777のグラフィック機能のなかで最も特徴的なのが，このDRAWコマンドでしょう．DRAWコマンドはタートル・グラフィックスの機能を持ち，サブコマンド列の指示どおりに図形を描くものです．タートル・グラフィックスと言えば，本書のLogoの章で詳しく触れていますが，もともとLogoやPascalのグラフィックス手法に用いられてきたもので，BASICには本来備わってはいない機能です．タートル・グラフィックスの概念は（Logoの章と重複しますが），カメ(turtle)が画面上を動き回ったとき，その軌跡を描画に使おうというもので，命令の基本となるのはカメの現在位置（相対座標）に対して位置及び角度を与えます．ですから普通のBASICのグラフィックスとは大幅に異なるもので，ちょっと頭の切り換えをしてから，以下の説明に目を通していただきたいと考えます．

DRAWコマンドの書式は，DRAW〈サブコマンド列〉で，表6-2に一覧表にしたサブコマンド

コマンド		形　式	意	味
状況を設定するコマンド（小文字で書いても結果は同じ）	A	An A'n	Anでは基準の座標軸(0°)に対する角度を指定 A'nでは現在設定されている座標軸に対する角度を指定 nは−32768〜32767の整数．初期設定はA0	angle 
	S	Sn	線を描くときの１単位のドット数を指定 nは1〜640の整数，初期設定はS4	scale
	C	Cn	描く線の色をカラーコードで指定 nは0〜15の整数．初期設定はC7	color
	P	Pn_1, n_2	現地点を含み，かつカラーコードn_2の色の境界線で囲まれた部分をn_1の色で塗りつぶす	paint 
	F	Fi	画面上の現地点にフラグ(旗)を立てる iは１〜７の整数，iを省略すると１とみなす 初期状態では(０，０)に７つのフラグがセットされている	flag
線を引くためのコマンド（小文字にすると移動のみで線は描かない）	M	Mx, y	現地点から絶対座標(x，y)の位置まで線を引く xは０〜263，yは０〜199の整数	move 
	W	Wh, v	Aコマンドで角度を指定した座標系で，x方向，y方向へそれぞれh，vだけはなれた地点まで線を引く h，vの単位はSコマンドで指定したドット数	walk 
	U D L R	Un Dn Ln Rn	現地点から，Aコマンドで指定した座標系で，それぞれ上下左右にnだけ離れた地点まで線を引く nの単位はSコマンドで指定したドット数．nを省略すると１とみなす	up down left right 
	G	Gi	現地点から，Fiコマンドでフラグを立てた点まで線を引く iは０〜７の整数 G0では現地点に点を打つ g0では何もしない	go to flag 
図形を描くためのコマンド	B	BEh, v BFh, v	現地点と，Aコマンドで指定した座標系のX，Y方向にそれぞれh，vだけ離れた点を結ぶ線を対角線とする四角を描く h，vの単位はSコマンドで指定したドット数 BEでは輪郭線のみ．BFでは中を塗りつぶした四角形	box empty box full 
	O	Or	y方向のrを半径とし，現地点を中心とした円を描く．x方向の値は自動的に補正され，真円を描く rの単位はSコマンドで指定したドット数	circle 
その他	<nサブコマンド列>		nに続くサブコマンド列を，n回繰り返して実行する	
	^M		音楽演奏サブコマンドへ移行．これ以降の文字列はPLAY文のサブコマンドとして解釈される	

An，Sn，Cnの指定を省略すると，それまでの各設定値を保持する

表6-2　グラフィックサブコマンド一覧

の示すとおりを実行します. DRAW が Logo などのタートル・グラフィックスと異なるのは，カメの現在位置で座標系を作り，x，yで与えられる位置に自由に移動させることが可能であること．また BOXE，BOXF，CIRCLE に相等する BE，BF，O というサブコマンドを持ち，一般に使われるのと同じ機能を持たせているのも，少々変ったところです．いわば，Logo のタートル・グラフィックスのエッセンスをそのままに，BASIC のグラフィックス・コマンドを取り込んだものと考えてよいのではないでしょうか．

さて，DRAW コマンドを説明してゆくにあたり，まずこのコマンドをリアルタイムで使うことのできるプログラムを紹介しておきましょう．Logo のタートル・グラフィックス同様，絶対座標の原点を画面のほぼ中央に定めるため，GLOCATE で座標の初期設定を行います．このリスト6-7を実行すると，画面の左上にメッセージ，SUB COMMAND ? が表示され，入力待ちの状態になります．以下，このプログラムを用いてサブコマンドを説明してゆきましょう．

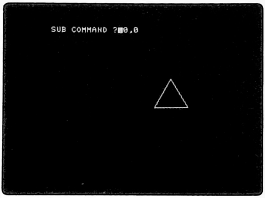
List 6-7

```
10 '-- learn draw --
20 WIPE
30 GLOCATE(160,100),1,1
40 WHILE 1
50   LOCATE(0,0)
60   INPUT /L 'SUB COMMAND ?',N$
70   IF N$='Gclear' THEN GCLEAR ELSE DRAW N$
80 WEND
```

最初は三角形を描いてみます．DRAW のサブコマンドは実に豊富で，三角形などいく通りにも描けるため，代表的なもので考えます．まずは絶対座標で位置を示して描く方法から．

SUB COMMAND ? M40，0

これは，M に続く40，0が絶対座標を表すもので，M は現在地から指定の絶対座標位置まで線を引く働きを持っています．画面中央から右側にかけて，描線されていますか？　次いで，



サブコマンドで「M40，0」，「M20，−35」，「M0，0」を実行した結果

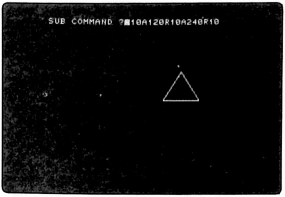
M20, −35

と入力してください．そして仕上げに，

M0, 0

で，三角形の完成です．このように絶対座標を使って描画してゆくものに対し，Logoのタートル・グラフィックスと同じような方法で三角形を描く場合は，

SUB COMMAND ? R10A120R10A240R10



サブコマンドで「R10A120R10A240R10」を実行した結果

となります．ここで用いたRは，カメの向き（X方向，初期設定では右向き）に指定ドット分線を引くサブコマンドで，この場合，1単位のドット数を定めるサブコマンドSの初期値が4になっているため，R10で40ドット分進みます．またAの方は基準座標値に対して角度を指定するもので，これがA'になると，今度はカメの向き（現在設定されている向き）に対する角度を変えることになります．ですからこの場合，最初のA120はA'でもカメの動きは変りませんが，次のA240のときは，A'を用いるならばA'120になります．さらに，この三角形の描線に色を付けたいときは，描線色指定のサブコマンドCを使い，さらに全部をくくって，

SUB COMMAND ? C 4 〈3 R 10 A' 120〉

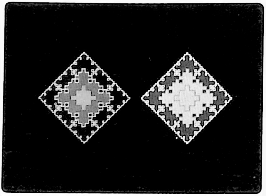
とすることも可能です．ここではカメの方向をX方向として考えてきましたが，Xの逆方向（L），Y方向（U），Yの逆方向（D）のコマンドを使うこともでき，90度単位の回転などに，思うとおりに線を伸ばせることから威力を発揮することでしょう．

これら以外にも数あるDRAWサブコマンドは，みなそれなりのポテンシャルを持っています．片っ端からリスト6-7のCOMMAND TESTで機能を試してみて，応用してみることが望ましいと思います（そのために，わざわざこのプログラムを作ったのですから！）．

参考のため，リスト6-8にクロスステッチを描くプログラムを例示します．一見してわかるよ

List 6-8

```
10 ´-- Cross stitch --
20  WIPE : DRAW 'S2' : CO=7
30  GCOLOR ,,15 : GCLEAR 15
40  PX=60 : PY=100 : ST=1 : GOSUB *COLOR
50  PX=180 : PY=100 : ST=1 : GOSUB *COLOR
60 END
70 *COLOR
80  FOR ST=1 TO 3
90    A=4-ST
100   CO=CO-1 :PCO=15-CO
110   GOSUB *DM
120  NEXT
130 RETURN
140 ´-- Line draw --
150 *L
160  C=C+1
170  IF A<>0 THEN GOTO *P1     ELSE GOTO *P2
180 *P1
190  A=A-1 : GOSUB *L
200  DRAW 'A´-90' : GOSUB *L
210  DRAW 'A´90' : GOSUB *L
220  DRAW 'A´90' : GOSUB *L
230  DRAW 'A´-90' : GOSUB *L
240  A=A+1
250  RETURN
260 ´
270 *P2
280  DRAW 'R(ST)'
290  RETURN
300 ´
310 ´-- Draw mainroutine --
320 *DM
330  L=(3^A)*ST
340  X=PX-L : Y=PY+L
350  DRAW 'm(X),(Y)A0C(CO)'
360  FOR I=1 TO 4
370    GOSUB *L
380    DRAW 'A´90'
390  NEXT
400  DRAW 'A45r5P(PCO),(CO)'
410  RETURN
```



クロスステッチを描く

うに，プログラムの大半がDRAW系のコマンドによってできており，DRAWコマンドがほかの表示コマンドのすべてをサポートしているからこそ，このようなプログラムが成立することが理解できるでしょう．

(6) その他のコマンド

これまでの直線や円を描くことで作画の基盤をなしたコマンドに対し，GGETとGPUTという，ドットの情報を配列に読み書きするコマンドを紹介してみましょう．

GGET $(x_1, y_1) - (x_2, y_2)$，A%(0)は(x_1, y_1)，(x_2, y_2)によって与えられる範囲のデータを，整数型配列A%に取り込むコマンドで，GPUTは，その配列データを任意の場所に表示します（同じように(x_1, y_1)，(x_2, y_2)で位置を指定します）．この2つのコマンドを用いることで，画面上の任意のパターンを指定の座標にコピーすることが可能となります．使い方は，GGET，GPUT双方とも，範囲を決定する対角の座標(x_1, y_1)，(x_2, y_2)と，その範囲に見合った大きさの配列（標準モードでドット数の1/4，高分解能モードでドット数の1/8）A%(0)をパラメータとして与えます．(x_1, y_1)は省略が可能で，前のグラフィックスコマンドで指定された最終値か，DRAWの現在地点が与えられます．また，GPUTには論理演算コードLと，透明色コードTを指定でき，透明色コードには表示したくない色のコードを与えます．

リスト6-9にGGET，GPUTを使ったプログラム例を紹介します．これは画面上のふたつのパターンをそれぞれ配列A%，B%に取り込んで，同じ位置に論理演算させて表示するものです．

コード	論理演算
0	論理演算なし（指定の色をそのまま表示）
1	すでに表示されている色のNOT（指定された色は無視される）
2	AND
3	OR
4	XOR

表6-3 論理演算コード

GGET，GPUT文を使うときは，さきに説明したように配列変数の大きさを宣言してやらなければいけません．配列変数をこの大きさで取ってくれ，とお願いするコマンドがDIMです．ここでは，GGET文で取り込む画面の座標値が縦36ドット，横36ドットなので，

$$36 \times 36 \div 4 = 324$$

となり324個の配列要素が必要なわけで，配列の添字は0から始まりますから，DIM A%(323)としておきます．

このプログラムでは，GGET，GPUT命令の使用法と共に，カラーコードの論理演算の仕組み

がビジュアルに理解できるようになっています．実行結果のように，ふたつの色の異なる四角形の重複部分の色の違いが一目でわかりますので，行番号40と60のBOXFのカラーコードを変えて色々試してみるとおもしろいでしょう．

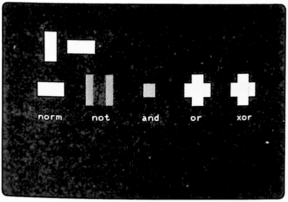
List 6-9

```
 10 '-- get & put --
 20 WIPE : CURSOR OFF
 30 DIM A%(323),B%(323)
 40 BOXF(10,0)-(25,35),3
 50 GGET (0,0)-(35,35),A%(0)
 60 BOXF(40,10)-(75,25),5
 70 GGET (40,0)-(75,35),B%(0)
 80 PRINT (0,12);"norm     not     and     or      xor":PRINT
 90 FOR I=0 TO 4
100   GPUT(I*64,50)-(I*64+35,85),A%(0)
110 GPUT(I*64,50)-(I*64+35,85),B%(0),I
120 NEXT
```

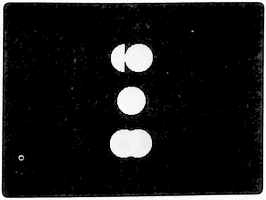
次にGPUTに透明色を指定したときの例をリスト6-10に示します．これを実行すると右上の画面のようになります．白く塗りつぶされた円を100行でGGETして，140，150行で画面の上の部分，190，200行で下の部分を描いているわけです．白い円の周りは黒で，200行ではこの黒が透明色に指定されています．このように透明色に指定された色は，文字どおり透明とみなされて，そのドットは全く描き込まれないのです．

List 6-10

```
 10 '
 20 '--transparent color--
 30 '
 40 WIPE : GMODE 1
 50 DEF INT A-Z
 60 DIM D(INT(41*41/4))
 70 GLOCATE(160,100),1,-1
 80 CIRCLE(0,0),20
 90 PAINT(0,0)
100 GGET(-20,20)-(20,-20),D(0)
110 '
120 'put normally
130 '
140 GPUT(-30,70)-(10,30),D(0)
150 GPUT(-10,70)-(30,30),D(0)
160 '
170 'put specifying transparent color
180 '
190 GPUT(-30,-30)-(10,-70),D(0)
200 GPUT(-10,-30)-(30,-70),D(0),0,0
```



リスト6-9の実行結果


リスト6-10の実行結果

6-4 カラーパレット・テクニック

標準状態のSMC-777では，カラージェネレータによって定められた16色の表示が可能です．しかし，この777にオプションのカラーパレットボード(SMI-733)を装着することで，RGB各色に対して16段階のうち任意の色調を設定できることから，16の3乗で都合4096種類(！)の色を出すことができるようになります．もちろん一度に表示できる色は16色までですから，4096色のうち任意の16色を選んで作画することになります．しかしこれだけ選択肢が広くなると，同系色を選ぶことで図形に微妙な陰影を付けて立体感を与えることができ，二次元上で立体的な作画をする上でのこの上ない武器となります．また，パレットによる色指定の切換えが瞬時に実行されることから，見かけ上の動画を作ることすら可能となります．もともとSMC-777はグラフィックスに強いマシンで，前節で紹介したグラフィックス・コマンドの充実ぶりにその一端を知ることができるのですが，このカラーパレットを戦列に加えることで，一層の機能充実が望めます．本節は，このパレットのテクニックを中心にグラフィックスを試みます．

(1) カラーパレットの基本

SMC-777のBASICでは，カラーパレットによってカラーコードに任意の色を定義するため，DEF PALETTE命令を用います．この命令は，

```
DEF PALETTE C, R, G, B
```

の書式で使われるもので，変数Cには定義色のカラーコードを与え，以下のRGBそれぞれに，任意のカラーコードで表現したい色を作るための，赤，緑，青の輝度を与えます．カラーコードは0～15で指定され，またRGBにはそれぞれ0～15の数値が与えられ，このとき，値が大きくなるほど輝度は高くなります．たとえばカラーコード0に青を定義したいときは，

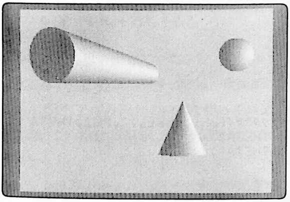
```
DEF PALETTE 0, 0, 0, 15
```

た，グラデーションを積極的に活用したものと言えるでしょう．前者が同系色のみでのグラデーションであったのに対し，後者は異なる色をグラデーションによってつないで見せるところに特徴があります．

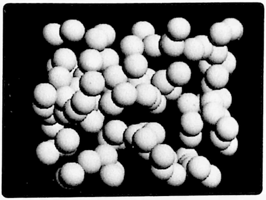
次に，グラデーションの特性を利用して，図形に影を与えて立体感を出す方法を考えてみます．リスト6-14がその例で，普通用いられる立体図形の手法である透視図(ワイヤーフレームによる)ではなく，曲面を持つものに対し陰影を付けて立体感をかもし出そうとしたものです．プログラムの構成は，行番号20～80が初期設定，100～200が円柱，220～300が球，そして320～390が円すいを描くプログラムになっており，非常に簡単な手順で立体感を生んでいます．

List 6-14

```
 10 '-- 3D-DEMONSTRATION --
 20  CONSOLE 40 : CURSOR OFF
 30  WIPE
 40  FOR I=0 TO 15
 50    DEF PALETTE I,0,I,15
 60  NEXT
 70  GCLEAR 9
 80  RAD=PI/180
 90 ' -- ｴﾝﾁｭｳ --
100  K1=40 : L1=50 : R1=30 : R2=27.5 : K2=160 : L2=70 : R3=10 : R4=9
110  C=-1
120  FOR I=0 TO 179
130    X1=R2*SIN(I*RAD)+K1 : X2=R4*SIN(I*RAD)+K2
140    Y1=L1-R1*COS(I*RAD) : Y2=L2-R3*COS(I*RAD)
150    IF I>90 THEN IF (I MOD 6)=0 THEN C=C-1
160    IF I<=90 THEN IF (I MOD 6)=0 THEN C=C+1
170    LINE(X2,Y2)-(X1,Y1),C,0
180  NEXT I
190  CIRCLE (K1,L1),R2,R1,1
200  PAINT (K1,L1),1,1
210 ' -- ｷｭｳ --
220  R1=20 : WX=270 : WY=50 : C=-1
230  FOR I=-90 TO 89
240    IF I<0 THEN C=0
250    IF I<=90 THEN IF (I MOD 6)=0 THEN C=C+1
260    Z=INT(SIN(I*RAD)*R1+.5) : Z0=COS(I*RAD)*R1
270    Z1=SIN(30*RAD)*Z0 : Z2=COS(30*RAD)*Z0
280    CIRCLE(WX+Z,WY),Z1,Z2,C,0
290    PAINT (WX+Z,WY),C,C
300  NEXT I
310 ' -- ｴﾝｽｲ --
320  C=9
330  FOR I=90 TO 260
340    IF I>120  THEN IF (I MOD 6)=0 THEN C=C-1
350    IF I<=120 THEN IF (I MOD 6)=0 THEN C=C+1
360    IF I>210  THEN IF (I MOD 6)=0 THEN C=0
370    X=SIN(I*RAD)*30 : Y=COS(I*RAD)*3
380    LINE (200,100)-(200+X,156-Y),C,0
390  NEXT
```



直感による立体図



計算による立体図

このリスト6-14のようなプログラムでは，影をどう付けたら立体感を得られるか，という問題に，直感的にこの程度だろうと判断し，半ば試行錯誤的にプログラムを修正していって立体感を出すよう工夫するといった方法です．もちろんこうした直感に頼る（あとの修正が面倒だが）方法でも何とかなりはするのですが，しかし数学的な整合性を持って陰影のあり方を考え，論理的に立体感を得た方がはるかに上策であることは論を待ちません．

陰影を付ける，つまり面処理による立体図形（ソリッドモデル）は，最先端のCG(コンピュータ・グラフィックス)でよく用いられる手法ですが，計算に非常に時間がかかり，8ビットのマイコンではちょっとオーバーワークの傾向がありますが，ここでは，比較的簡単な球体のソリッドモデルを紹介しておきましょう．

リスト6-15に紹介するのが，そうした論理的な手法の実践例です．このプログラムを実行して，描く球の画面縦方向の半径と光源の位置とを入力すると，指定した球を100個GPUTで描きます．光源の位置の指定は図6-11のように2つの角度ϕとθで行います．

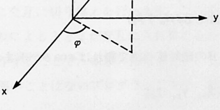


図6-11　光源の方向の指定

List 6-15

```
10 '-- ｹｲｻﾝﾆ ﾖﾙ ｷｭｳ --
20  WIPE
30  CONSOLE 80
40  DEF FNRAD(DEG%)=PI*DEG%/180
50  INPUT "yｼﾞｸ ﾎｳｺｳ ﾉ ﾊﾝｹｲ";R
60  RX=INT(R*1.25)+1
70  INPUT "ｺｳｹﾞﾝ ﾉ ｲﾁ ｦ ﾆｭｳﾘｮｸ ｼﾃｸﾀﾞｻｲ(phi,theta)",PHI,THETA
80  LX=COS(FNRAD(PHI))*SIN(FNRAD(THETA))
90  LY=SIN(FNRAD(PHI))*SIN(FNRAD(THETA))
100 LZ=-COS(FNRAD(THETA))
110 CONSOLE 40
120 PRINT (5,12);"Now computing.  Please wait."
130 PRINT (0,17);SPACE$(R*2-1);"goal"
140 LOCATE(0,17)
150 DIM PD%(INT((RX*2-1)*(R*2-1)/4))
160 FOR Z=-R+1 TO R-1
170   X1=SQR(R*R-Z*Z)
180   Y1=INT(X1*1.25)
190   FOR Y2=-Y1 TO Y1
200     Y3=Y2/1.25
210     X2=SQR(X1*X1-Y3*Y3)
220     CL=(X2*LX+Y3*LY+Z*LZ)/R*14
230     IF CL<0 THEN CL=1 ELSE CL=CL+1
240     P%=(Z+R-1)*(RX*2-1)+Y2+RX-1
250     PD%(P%¥4)=PD%(P%¥4) OR CADR(CINT(CL)*(16^(P% MOD 4)))
260   NEXT
270   PRINT ">";
280 NEXT
290 CCLEAR
300 FOR I=1 TO 15
310   DEF PALETTE I,0,0,I
320 NEXT
330 CURSOR OFF
340 FOR I=1 TO 100
350   X=IRND(321-RX*2) : Y=IRND(201-R*2)
360   GPUT(X,Y)-(X+RX*2-2,Y+R*2-2),PD%(0),,0
370 NEXT
380 EVAL INKEY$(1)
390 CONSOLE 80
400 END
```

このプログラムの計算に当たり，最も単純に，球には平行光線が当たるとし，それぞれの点の明るさは，光線の入射角の $\cos\theta$ に比例するとして計算しています．ベクトルA，Bの内積は，

$$A \cdot B = |A||B|\cos\theta \quad (\theta \text{ はA, Bのなす角})$$

ですから，A，Bの内積をA，Bの絶対値の積で割れば $\cos\theta$ が求まるわけです．また，

$$A=(X_1,\ Y_1,\ Z_1),\ B=(X_2,\ Y_2,\ Z_2)$$

とすると，

$A \cdot B = X_1X_2 + Y_1Y_2 + Z_1Z_2$

であることも思い出してください.

ところで，このプログラムでは一度画面に描いてからGGETしてGPUTするのではなく，始めからGPUTする配列の内容を作っています．250行でGPUTする配列の内容を書き込んでいますが，この中にCADRという関数が出てきます．この関数は符号無し2進数を符号付き2進数に変えるもので，32768～65535を渡すと，－32768～－1を返します．また半径10の球の計算をするにも1分以上かかり，精神衛生上の配慮から，ドットを1行分計算し終るごとに［>］を表示して，これがgoalのところまで達したら計算終了であるようにしてあります．

2つの球が重なった部分をうまく描くために，360行のGPUTには透明色が指定してあり，また，球は一番暗く見えるところまでも背景と同じ色ではなくしてあります．

(3) 動画

カラーパレットを用いて色の指定を変えると，すでに表示されている色が瞬間的に切り換えられます．これを利用して，表示されているカラーコードにバックグラウンドと同じ色を指定したり，また再びもとの色を指定することにより，見かけの上での動画を作ることができます．

リスト6-16に示すのが，カラーパレットの切り換えを使って回転を見せかけるプログラムです．このプログラムは，8つに分割した円のそれぞれのパートを青と緑に塗り分け，交互に青と緑に色を変化させることであたかも青色の部分が回転しているように見せる水車です．

このプログラムの主要な部分は，行番号200から250の間です．160行で，青と緑を塗り分けていますが，ここで青と緑のカラーコードを1と2に設定しています．

(I MOD 2)＋1

MODというのは，整数剰余，つまり割り算の余りを返す関数です．ここでは，Iには偶数か奇数が代入されるので，それを2で割った余りに1を足した数，つまり1か2が得られます．次に，220行，230行の，

((I＋1) MOD 2)＋1，0，15，0
(I MOD 2)＋1，0，0，15

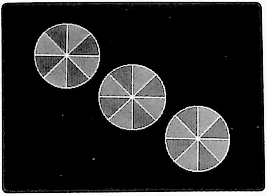
により，青を緑に，緑を青に交互に切り換えを行います．ここでは，(I+1)とIを2で割った余りに1を足すことで，偶奇数による1と2が交互に入れ替わるわけです．また，240行のPAUSEによるタイマーの数字を操作することで，水車の回転スピードを自由に変化させることができます．まァ，見ている分には何のことはないプログラムですが，動画を作る上での参考にはなると思います．

List 6-16

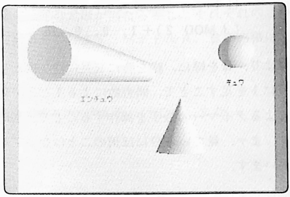
```
10 '-- movement by palette --
20  WIPE
30  CURSOR OFF
40  FOR J=1 TO 3
50    X=J*100-50 : Y=J*50
60    CIRCLE(X,Y),50,,7
70    FOR I=0 TO 360 STEP 45
80      TH=I*PI/180
90      X1=X+50*COS(TH)
100     Y1=Y+50*SIN(TH)*.8
110     LINE(X,Y)-(X1,Y1),7
120   NEXT I
130   FOR I=1 TO 8
140     X1=X+40*COS((I*45-5)*PI/180)
150     Y1=Y+40*SIN((I*45-5)*PI/180)*.8
160     PAINT(X1,Y1),(I MOD 2)+1,7
170   NEXT I
180  NEXT J
190 '
200  WHILE 1
210    I=I+1
220    DEF PALETTE ((I+1) MOD 2)+1,0,15,0
230    DEF PALETTE (I MOD 2)+1,0,0,15
240    PAUSE 5
250  WEND
```

(4) 24色表示

カラーパレットを使っても，カラーコードは０～15までしかありませんので，一度に表示可能な色は16色どまり．しかし，カラーパレットの色を使用するか，カラージェネレータの色を使用するかの切り換えを，図6-12に示したように51Hポートに出力するデータでキャラクタとグラフィックスを別々に行えるので，たとえばキャラクタをカラージェネレータの色，グラフィックスをカラーパレットの色に指定し，カラーパレットにカラージェネレータで決定されているのとは別の色を指定することで，キャラクタ８色＋グラフィックス16色の合計24色を一度に表示でき



パレットを利用した水車



24色表示を利用した画面

るようになります．ただし，カラージェネレータでは表示される色が決ってしまっているため，24色の全部を任意の色ということはできず，カラージェネレータで決まっている８色（キャラクタ）＋任意の16色（グラフィックス）という方法か，またはカラージェネレータで決っている16色（グラフィックス）＋任意の８色（キャラクタ）という組合せになります．しかし，たとえばグラデーションで描いているときなどに，16色の全部を同系色に指定してしまっても，キャラクタをカラージェネレータの色（グラデーションに用いたのとは別系統の色）で表示すれば，メッセージの表示など非常に鮮明になりますし，キャラクタを上手くグラフィックスに組み合わせて使用すれば，よりカラフルな図形を描くことも可能であると考えられます．

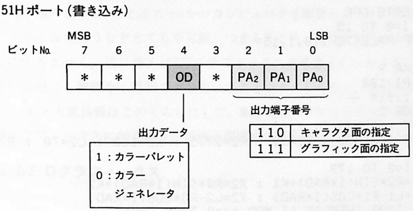


図6-12　カラーパレット/カラージェネレータの切り換え

この方法を実際に使用するときは，51H ポートに適当なデータを出力すればよいわけですから，たとえばキャラクタをカラージェネレータの色で表示したいときは，BASIC の OUT 命令で，

```
OUT &H51, &H06
```

を実行すれば切り換わります．しかし，こんなメンドウなことをするまでもなく，777 BASIC には上記に対応するコマンドがあり，CPALETTE と GPALETTE を用いることでも同等の結果となり，実行方法は，

CPALETTE ON　：キャラクタをカラーパレットの色で表示
CPALETTE OFF ：キャラクタをカラージェネレータの色で表示
GPALETTE ON　：グラフィックスをカラーパレットの色で表示
GPALETTE OFF ：グラフィックスをカラージェネレータの色で表示

と指定します．これは BASIC のマニュアルには（どうしたわけだか）記載されていない方法ですが，BASIC をディスアセンブルしてみると，予約語と中間コードの対応表らしきものが見付かりますし，そのなかにこの２つのコマンドも含まれていて，いじくり回しているうちに使い方がわ

かったという次第. 前掲のリスト6-14にこのCPALETTE OFFを組み込んで, メッセージを表示できるように加工したのがリスト6-17です. 24色という既存の色より8色も多いグラフィックスは, SMC-777のグラフィックス機能をより高いものにしてくれるものと思います.

パレットを用いたグラデーションや動画の作画は, なによりパレットの機能を適確に把握し, あとはひたすら慣れること, つまり試行錯誤が上達の早道. ちなみに, SMC-777のデモも, パレットを使って描いたものでした…….

List 6-17

```
 10 '-- 3D-DEMONSTRATION --
 20  CONSOLE 40 : CURSOR OFF
 30  WIPE
 35  CPALETTE OFF
 40  FOR I=0 TO 15
 50    DEF PALETTE I,0,I,15
 60  NEXT
 70  GCLEAR 9
 80  RAD=PI/180
 90 ' -- ｴﾝﾁｭｳ --
 95  CCOLOR 2
 96  PRINT (10,12);'ｴﾝﾁｭｳ'
100  K1=40 : L1=50 : R1=30 : R2=27.5 : K2=160 : L2=70 : R3=10 : R4=9
110  C=-1
120  FOR I=0 TO 179
130    X1=R2*SIN(I*RAD)+K1 : X2=R4*SIN(I*RAD)+K2
140    Y1=L1-R1*COS(I*RAD) : Y2=L2-R3*COS(I*RAD)
150    IF I>90 THEN IF (I MOD 6)=0 THEN C=C-1
160    IF I<=90 THEN IF (I MOD 6)=0 THEN C=C+1
170    LINE(X2,Y2)-(X1,Y1),C,0
180  NEXT I
190  CIRCLE (K1,L1),R2,R1,1
200  PAINT (K1,L1),1,1
210 ' -- ｷｭｳ --
215  CCOLOR 4
216  PRINT (32,10);'ｷｭｳ'
220  R1=20 : WX=270 : WY=50 : C=-1
230  FOR I=-90 TO 89
240    IF I<0 THEN C=0
250    IF I<=90 THEN IF (I MOD 6)=0 THEN C=C+1
260    Z=INT(SIN(I*RAD)*R1+.5) : Z0=COS(I*RAD)*R1
270    Z1=SIN(30*RAD)*Z0 : Z2=COS(30*RAD)*Z0
280    CIRCLE(WX+Z,WY),Z1,Z2,C,0
290    PAINT (WX+Z,WY),C,C
300  NEXT I
310 ' -- ｴﾝｽｲ --
315  CCOLOR 6
316  PRINT (24,21);'ｴﾝｽｲ'
320  C=9
330  FOR I=90 TO 260
340    IF I>120  THEN IF (I MOD 6)=0 THEN C=C-1
350    IF I<=120 THEN IF (I MOD 6)=0 THEN C=C+1
360    IF I>210  THEN IF (I MOD 6)=0 THEN C=0
370    X=SIN(I*RAD)*30 : Y=COS(I*RAD)*3
380    LINE (200,100)-(200+X,156-Y),C,0
390  NEXT
```

6-5　グラフィックス上級編

近年のコンピュータ・グラフィックス（CG）の発達には，目を見はらせるものがあります．写真とも絵画ともつかぬ微妙な階調を持ったグラフィックスであるとか，面を定義して色を付け，べら棒な時間を費して回転をさせてみたりとか――なかにはソニーのウォークマンのCFの如く，スーパーコンピュータ，CRAY-1を使ったグラフィックスまで登場して，まさに留まるところを知らぬ勢いです．パソコンの分野におけるグラフィックスにしても，かなり程度の高いものがずい分と出現するようになっています．計算機をまるで知らない人であれば，超大型機を使った3-Dの回転像などに仰天して，たかがパソコンに何ができる，などとケチをつけたりもします．が，すでに御存知のように，たとえスーパー・コンピュータを駆使しようと，精密な画面の回転や着色をリアルタイムで行うなどとても不可能．つまるところ，解像度と少々の演算速度の速さに頼っているにすぎません．逆に言えば777グラフィックスで基礎をしっかり身に付けておけば，かなり高度な応用にも無理なく移行できるということです（もちろん，当人の努力と才能があっての話だが）．まァ大風呂敷はこのくらいにして，本節ではグラフィックスでも程度が高いとされている3-Dグラフィックスに的を絞り，線画と面画でこれを試みてみましょう．

（1）線画による3-Dグラフィックス

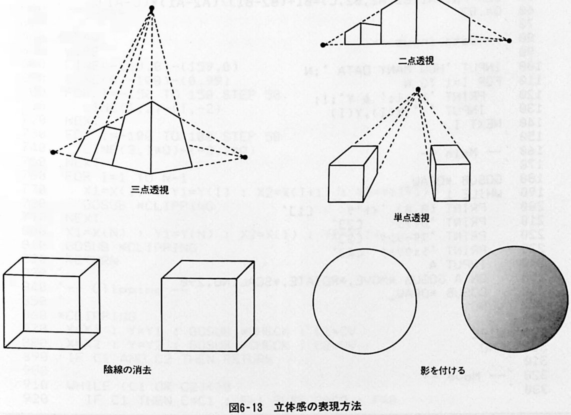


図6-13　立体感の表現方法

3次元に住むわれわれ人間の常として，目に映るものを平面に表現するときに，何とか立体的に見えるよう工夫するものです．試しに幼児にクレヨンと画用紙を与えてみると，線画から着色をやっつけて，最後は遠近感を出そうとあれこれやってみるはずです．

立体感，遠近感を表現するとき，通常は図6－13に示すように，影を付けて立体感を得たり，陰線を消去したり，また透視変換で遠近感を出してみたりします．コンピュータを用いて立体作画をする場合も，このような方法を踏襲します．しかし，単に紙に描くわけではないため，作図のためのデータ（座標データ）を計算で変換させることにより，図形を移動させたり回転させて表示することになります．トロンに代表されるCGを用いた映画などでもこのテクニックが用いられているわけで，コンピュータらしい作画ができる部分です．まァ，SMC－777とCRAY－1に同じ仕事をさせるわけにもいきませんので，ここでは基本的なところを扱ってみることにしましょう．

2次元図形の変換

List 6-18

```
 10 '
 20 '================== 2D graphic package ====================
 30 '
 40  WIPE : CONSOLE 80 : GLOCATE (160,100),,-1
 50  DEF FNY(A1,B1,A2,B2,C)=B1+(B2-B1)/(A2-A1)*(C-A1)
 60  Q=.82
 70 '
 80 '-- Data input --
 90 '
100  INPUT "HOW MANY DATA ";N
110  FOR I=1 TO N
120    PRINT "X";I;" & Y";I;
130    INPUT " ";X(I),Y(I)
140  NEXT I
150 '
160 '-- Main --
170 '
180  GOSUB *DRAW_
190  WHILE 1
200    PRINT (0,0) "ｲﾄﾞｳ     [1]"
210    PRINT "ｶｲﾃﾝ      [2]"
220    PRINT "ｽｹｰﾘﾝｸﾞ   [3]"
230    PRINT "ｼｭｳﾘｮｳ    [4]"
240    INPUT A
250    ON A GOSUB *MOVE,*ROTATE,*SCALING,290
260    GOSUB *DRAW_
270  WEND
280 '
290  WIPE
300  END
310 '
320 '-- Move --
330 '
```

```
340 *MOVE
350  INPUT "イドウド X & Y ";TX,TY
360  FOR I=1 TO N
370    X(I)=X(I)+TX
380    Y(I)=Y(I)+TY
390  NEXT
400  RETURN
410 '
420 '-- Rotate --
430 '
440 *ROTATE
450  INPUT "カクド ";TH
460  TH=TH*PI/180
470  FOR I=1 TO N
480    X=X(I)*COS(TH)-Y(I)*SIN(TH)
490    Y=X(I)*SIN(TH)+Y(I)*COS(TH)
500    X(I)=X : Y(I)=Y
510  NEXT
520  RETURN
530 '
540 '-- Scaling --
550 '
560 *SCALING
570  INPUT "ヒリツ X & Y ";SX,SY
580  FOR I=1 TO N
590    X(I)=X(I)*SX
600    Y(I)=Y(I)*SY
610  NEXT
620  RETURN
630 '
640 '-- Draw --
650 '
660 *DRAW_
670  WIPE
680  LINE(-160,0)-(159,0)
690  LINE(0,-100)-(0,99)
700  FOR I=-150 TO 150 STEP 50
710    LINE(I,2)-(I,-2)
720  NEXT
730  FOR I=-100 TO 100 STEP 50
740    LINE(3,I*Q)-(-3,I*Q)
750  NEXT
760  FOR I=1 TO N-1
770    X1=X(I) : Y1=Y(I) : X2=X(I+1) : Y2=Y(I+1)
780    GOSUB *CLIPPING
790  NEXT
800  X1=X(N) : Y1=Y(N) : X2=X(1) : Y2=Y(1)
810  GOSUB *CLIPPING
820  RETURN
830 '
840 '-- Clipping --
850 '
860 *CLIPPING
870  X=X1 : Y=Y1 : GOSUB *CHECK : C1=CV
880  X=X2 : Y=Y2 : GOSUB *CHECK : C2=CV
890  IF C1 AND C2 THEN RETURN
900 '
910  WHILE (C1 OR C2)<>0
920    IF C1 THEN C=C1 : F=1 ELSE C=C2 : F=0
```

```
 930    IF (C AND 1)=0 THEN 970
 940      X=159 : Y=FNY(X1,Y1,X2,Y2,X)
 950      C=C AND 14
 960      GOTO 1070
 970    IF (C AND 2)=0 THEN 1010
 980      X=-160 : Y=FNY(X1,Y1,X2,Y2,X)
 990      C=C AND 13
1000      GOTO 1070
1010    IF (C AND 4)=0 THEN 1050
1020      Y=99/Q : X=FNY(Y1,X1,Y2,X2,Y)
1030      C=C AND 11
1040      GOTO 1070
1050    Y=-100/Q : X=FNY(Y1,X1,Y2,X2,Y)
1060    C=0
1070    IF F THEN X1=X : Y1=Y : C1=C ELSE X2=X : Y2=Y : C2=C
1080  WEND
1090  LINE(X1,Y1*Q)-(X2,Y2*Q),4
1100  RETURN
1110
1120 *CHECK
1130  CV=0
1140  IF X>159  THEN CV=1
1150  IF -160>X THEN CV=CV+2
1160  IF Y>99/Q THEN CV=CV+4
1170  IF -100/Q>Y THEN CV=CV+8
1180  RETURN
```

リスト6－18を見てください．2次元の図形を変換させるプログラムです．画面のほぼ中央を原点として，画面上の図形を様々に変換します．変換の種類は，

・平行移動（サブルーチン＊MOVE）

・回転（サブルーチン＊ROTATE）

・スケーリング（サブルーチン＊SCALING）

の3つがパッケージで用意されています．次に，基本図形として図6－14に示すような四角形を用い，変換してゆく手続きを説明してみましょう．

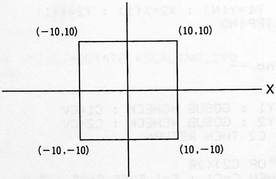


図6-14　基本図形

プログラムを実行すると，データの数を問いかけてきます．ここでは図6－14を基本図形とするために，四隅の座標データが必要なため，4を入力してやります．次いで各座標のデータを，

X 1 &Y 1 ? 10, −10
X 2 &Y 2 ? −10, −10
X 3 &Y 3 ? −10, 10
X 4 &Y 4 ? 10, 10

と与えてやり，実行することで図6－14と同じ図形が得られます．このとき，画面には変換のメニューが，

イドウ [1]
カイテン [2]
スケーリング [3]
シュウリョウ [4]

と表示されますので，まずは1を入力して平行移動から実行してみましょう．平行移動は図6－15に示すように点P（x，y）をP′（x′，y′）に移動させる手続きで，

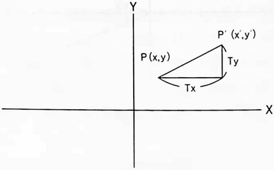
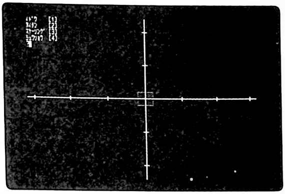
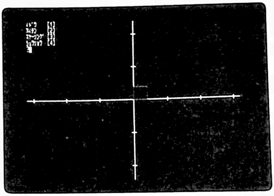


図6-15　平行移動



基本図形の表示



平行移動

$$\left.\begin{aligned} x' &= x + T_x \\ y' &= y + T_y \end{aligned}\right\} \text{式 1}$$

により得られます．このプログラムではx，yの移動量を聞いてきますから，たとえば10，10を入力してやると，サブルーチン＊MOVEで式1の手続きが四隅のデータに各々施され，該当座標まで移動させて表示します．次に行うのは2の回転，これを得るには，

$$\left.\begin{aligned} x' &= \cos\theta \times x - \sin\theta \times y \\ y' &= \sin\theta \times x + \cos\theta \times y \end{aligned}\right\} \text{式 2}$$

の式で表され，図6-16のような回転を行います．たとえば10，10の座標に平行移動した図形に，原点を中心に45度の回転を与える場合，回転角を聞かれますから，45と入力します．＊ROTATEでは，この入力値を弧度法に変換したあと，式2を実行させ，該当回転角を実施した図形を表示させます．

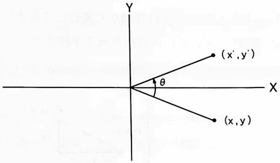


図6-16　回転

そして，最後のスケーリング．これは図形の拡大もしくは縮小を意味するもので，例えば図6-17に示すようなスケーリングは，

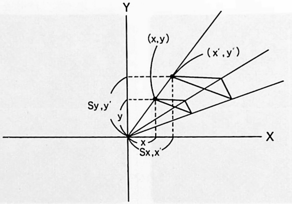
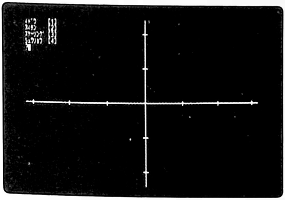


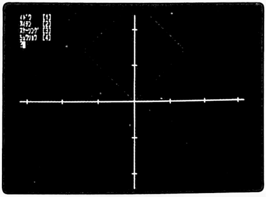
図6-17　スケーリング

$$\left.\begin{aligned} x' &= x \times S_x \\ y' &= y \times S_y \end{aligned}\right\} \text{式 3}$$

の式で得られます．メニューの3を選択し，x，yの比率に5，5を選択してみると，さきほどの回転で得られた図形が，x，yともに5倍に拡大されて得られます．



平行移動ののち回転



回転ののち拡大

以上が基本的な2次元図形の変換プログラムの説明になりますが，このなかで注意しなければならないのは，変換後のデータが画面の範囲外に出てしまう場合を考え，クリッピングという操作をする必要があることです．クリッピングは図6-18に示すよう，画面から図形がはみ出したときに，画面内の図形のみを囲み取る操作です．リスト6-18では＊CLIPPING以降がそのためのルーチンで，画面の枠を図形が横切るときの交点を求め，画面内のみを描くようにしているのです．また，変換式1～3は，一般的には表6-4に示すように，行列計算式によって表現されます．

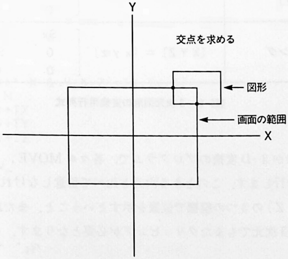


図6-18　2次元図形のクリッピング

変　換	行列の計算式
平行移動	$[X\ Y]=[x\ y]+[Tx\quad Ty]$
回　　転	$[X\ Y]=[x\ y]\begin{bmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{bmatrix}$
スケーリング	$[X\ Y]=[x\ y]\begin{bmatrix}Sx & 0\\ 0 & Sy\end{bmatrix}$

表6-4　2次元図形の変換用行列式

3次元図形の変換

3次元の図形を変換するときでも，基本的には2次元と同じ方法をとります．3次元のときは行列計算式が，表6-5のようなものに変り，少々複雑にはなりますが，変換自体は行列演算によって行うことができるのです．

変　換	行列の計算式
平行移動	$[X\ Y\ Z]=[x\ y\ z]+[\ Tx\quad Ty\quad Tz\ ]$
X軸回りの回転 (ピッチ)	$[X\ Y\ Z]=[x\ y\ z]\begin{bmatrix}1 & 0 & 0\\ 0 & \cos P & \sin P\\ 0 & -\sin P & \cos P\end{bmatrix}$
Y軸回りの回転 (ヘディング)	$[X\ Y\ Z]=[x\ y\ z]\begin{bmatrix}\cos H & 0 & -\sin H\\ 0 & 1 & 0\\ \sin H & 0 & \cos H\end{bmatrix}$
Z軸回りの回転 (バンク)	$[X\ Y\ Z]=[x\ y\ z]\begin{bmatrix}\cos B & \sin B & 0\\ -\sin B & \cos B & 0\\ 0 & 0 & 1\end{bmatrix}$
スケーリング	$[X\ Y\ Z]=[x\ y\ z]\begin{bmatrix}Sx & 0 & 0\\ 0 & Sy & 0\\ 0 & 0 & Sz\end{bmatrix}$

表6-5　3次元図形の変換用行列式

リスト6-19に示すのが3-D変換のプログラムで，各々＊MOVE，＊ROTATE 1～3，＊SCALINGで変換を実行します．このとき2次元と比べて留意しなければならないのは，3次元であるから(X，Y，Z)の3つの座標で位置を示すということ．また，2次元でクリッピングを行ったのと同様に，3次元でもまたクリッピングが必要となります．

List 6-19

```
10 '
20 '================== 3D graphic package ==================
30 '
40  WIPE : CONSOLE 80 : GLOCATE (160,100),,-1
50  D=2000 : PT=200
60  AX=PT/160 : AY=PT/100
70  BX=1/AX : BY=1/AY
80 '
90 '-- Data input --
100 '
110  INPUT "SAMPLE DATA (Y/N) ";A$
120  IF A$="Y" THEN N=11 : GOTO 140
130    INPUT "HOW MANY DATA ";N
140  DIM X(N),Y(N),Z(N),M(N)
150  FOR I=1 TO N
160    IF A$="Y" THEN READ X(I),Y(I),Z(I),M(I) : GOTO 190
170      PRINT "X";I;" & Y";I;" & Z";I;" M";I;
180      INPUT " ";X(I),Y(I),Z(I),M(I)
190  NEXT I
200  GOSUB *DRAW_
210 '
220 '-- Main --
230 '
240  WHILE 1
250    LOCATE (0,0)
260    PRINT  "ｲﾄﾞｳ          [1]"
270    PRINT  "ｶｲﾃﾝ X        [2]"
280    PRINT  "ｶｲﾃﾝ Y        [3]"
290    PRINT  "ｶｲﾃﾝ Z        [4]"
300    PRINT  "ｽｹｰﾘﾝｸﾞ       [5]"
310    PRINT  "ｼﾃﾝ ﾉ ﾍﾝｺｳ    [6]"
320    PRINT  "ｻﾞﾋｮｳ         [7]"
330    PRINT  "ｼｭｳﾘｮｳ        [8]"
340    INPUT A
350    ON A GOSUB *MOVE,*ROTATE1,*ROTATE2,*ROTATE3,*SCALING,*VIEW,
*COOR,380
360  WEND
370 '
380  END
390 '
400 '-- Move --
410 '
420 *MOVE
430  INPUT "ｲﾄﾞｳﾄﾞ X & Y & Z";TX,TY,TZ
440  FOR I=1 TO N
450    X(I)=X(I)+TX
460    Y(I)=Y(I)+TY
470    Z(I)=Z(I)+TZ
480  NEXT
490  GOSUB *DRAW_
500  RETURN
510 '
520 '-- Rotate 1 --
530 '
540 *ROTATE1
550  INPUT "ｶｸﾄﾞ ";P
560  P=P*PI/180
```

```
570  SINP=SIN(P)
580  COSP=COS(P)
590  FOR I=1 TO N
600     X= X(I)
610     Y=-Z(I)*SINP+Y(I)*COSP
620     Z= Y(I)*SINP+Z(I)*COSP
630     X(I)=X : Y(I)=Y : Z(I)=Z
640  NEXT
650  GOSUB *DRAW_
660  RETURN
670 '
680 '-- Rotate 2 --
690 '
700 *ROTATE2
710  INPUT "ｶｸﾄﾞ ";H
720  H=H*PI/180
730  SINH=SIN(H)
740  COSH=COS(H)
750  FOR I=1 TO N
760     X= X(I)*COSH+Z(I)*SINH
770     Y= Y(I)
780     Z=-X(I)*SINH+Z(I)*COSH
790     X(I)=X : Y(I)=Y : Z(I)=Z
800  NEXT
810  GOSUB *DRAW_
820  RETURN
830 '
840 '-- Rotate 3 --
850 '
860 *ROTATE3
870  INPUT "ｶｸﾄﾞ ";B
880  B=B*PI/180
890  SINB=SIN(B)
900  COSB=COS(B)
910  FOR I=1 TO N
920     X=X(I)*COSB-Y(I)*SINB
930     Y=X(I)*SINB+Y(I)*COSB
940     Z=Z(I)
950     X(I)=X : Y(I)=Y : Z(I)=Z
960  NEXT
970  GOSUB *DRAW_
980  RETURN
990 '
1000 '-- Scaling --
1010 '
1020 *SCALING
1030  INPUT "ﾋﾘﾂ X & Y & Z";SX,SY,SZ
1040  FOR I=1 TO N
1050     X(I)=X(I)*SX
1060     Y(I)=Y(I)*SY
1070     Z(I)=Z(I)*SZ
1080  NEXT I
1090  GOSUB *DRAW_
1100  RETURN
1110 '
1120 '-- View point --
1130 '
1140 *VIEW
```

```
1150  PRINT D,PT
1160  INPUT "ｼﾃﾝ & ﾊﾟｰｽ ";D,PT
1170  AX=PT/160 : AY=PT/100
1180  BX=1/AX : BY=1/AY
1190  GOSUB *DRAW_
1200  RETURN
1210 '
1220 '-- Coordinate --
1230 '
1240 *COOR
1250  PRINT "      X             Y             Z"
1260  FOR I=1 TO N
1270  PRINT USING "##  #####.####   #####.####   #####.####";I,X(I)
,Y(I),Z(I)
1280  IF (I MOD 20)=0 THEN PRINT "HIT ANY KEY TO CONTINUE" : EVAL
INKEY$(1)
1290  NEXT I
1300  PRINT "HIT ANY KEY " : EVAL INKEY$(1)
1310  CCLEAR
1320  RETURN
1330 '
1340 '-- Draw --
1350 '
1360 *DRAW_
1370  WIPE
1380  FOR I=1 TO N-1
1390    IF M(I+1)=0 THEN 1430
1400      X1=X(I)   : Y1=Y(I)   : Z1=Z(I)+D
1410      X2=X(I+1) : Y2=Y(I+1) : Z2=Z(I+1)+D
1420    GOSUB *CLIP
1430  NEXT I
1440  RETURN
1450   '
1460   '-- Clipping for 3D --
1470   '
1480   *CLIP
1490    X=X1 : Y=Y1 : Z=Z1 : GOSUB *CHECK : C1=CV
1500    X=X2 : Y=Y2 : Z=Z2 : GOSUB *CHECK : C2=CV
1510   '
1520    WHILE (C1 OR C2)<>0
1530      IF C1 AND C2 THEN RETURN
1540      IF C1 THEN C=C1 ELSE C=C2
1550     '
1560      IF C AND 1 THEN W=(Z1+X1*AX)/((X1-X2)*AX-(Z2-Z1)): GOTO 1610
1570      IF C AND 2 THEN W=(Z1-X1*AX)/((X2-X1)*AX-(Z2-Z1)): GOTO 1610
1580      IF C AND 4 THEN W=(Z1+Y1*AY)/((Y1-Y2)*AY-(Z2-Z1)): GOTO 1610
1590                      W=(Z1-Y1*AY)/((Y2-Y1)*AY-(Z2-Z1))
1600     '
1610      X=W*(X2-X1)+X1 : Y=W*(Y2-Y1)+Y1 : Z=W*(Z2-Z1)+Z1
1620     '
1630      IF C=C1 THEN X1=X : Y1=Y : Z1=Z : GOSUB *CHECK : C1=CV
               ELSE X2=X : Y2=Y : Z2=Z : GOSUB *CHECK : C2=CV
1640    WEND
1650   '
1660    X1=PT*X1/Z1 : Y1=PT*Y1/Z1
1670    X2=PT*X2/Z2 : Y2=PT*Y2/Z2
1680    LINE(X1,Y1)-(X2,Y2),2 : RETURN
1690   '
```

```
1700      *CHECK
1710       CV=0
1720       DZ=.00000000001*ABS(Z*BX)
1730       IF X<-Z*BX-DZ THEN CV=1
1740       IF X> Z*BX+DZ THEN CV=CV+2
1750       IF Y<-Z*BY-DZ THEN CV=CV+4
1760       IF Y> Z*BY+DZ THEN CV=CV+8
1770       RETURN
1780 '
1790 '-- Sample data --
1800 '
1810 DATA  1000,-500, 1000,0
1820 DATA  1000,-500,-1000,1
1830 DATA -1000,-500,-1000,1
1840 DATA -1000,-500, 1000,1
1850 DATA  1000,-500, 1000,1
1860 DATA     0, 500,    0,1
1870 DATA  1000,-500,-1000,1
1880 DATA     0, 500,    0,0
1890 DATA -1000,-500,-1000,1
1900 DATA     0, 500,    0,0
1910 DATA -1000,-500, 1000,1
```

3次元の場合，2次元とは異なって，図6-19に示すような，視野ピラミッドといったような考え方が必要となってきます．3次元の観点で物体を眺めるとき，遠くから見た場合は全体が見えたのに，近寄ると部分しか見えなくなるのは当然のこと．だから視野ピラミッドを想定して，見える範囲を限定するという作業が必要なわけで，プログラムの行番号1480～1770でそれを行っており，アルゴリズムは2次元の場合とほとんど変りません．

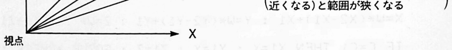


図6-19　視野ピラミッド

数値上での変換に際しての留意点は以上のようなものですが，実際に画面に表示するときは，3次元のデータを2次元に投影してやらねばなりません．そこで，図6-13に示したなかの透視変換を用いて平面上に投影するわけですが，このプログラムでは単点透視での投影を行います．こ

れは非常に単純な方法で，３次元物体の見かけ上の大きさは，視点からの距離に反比例することがはっきりしていますから，視点を原点に定めた場合，点（x，y，z）の見かけ上の位置（X，Y）を，

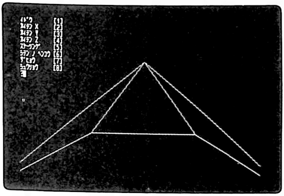
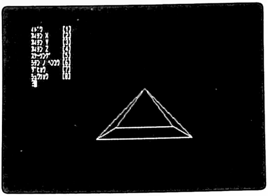
$$\left.\begin{array}{l} X = x/z \\ Y = y/z \end{array}\right\} \text{式 4}$$

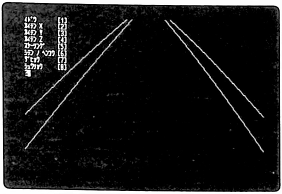
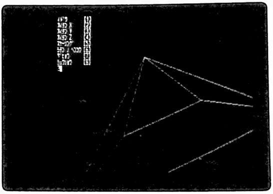
で投影してしまいます．リスト6－19のなかでは1660行，1670行がこの操作を担当しています．

リスト6－19の実行の手順を説明すると，基本的には２次元で行ったものと大差はないのですが，座標値をいくつも入力するのが面倒なのでサンプルデータを入れてありますから，

SAMPLE DATA　(Y／N)

と聞かれたときに，Ｙを入力することで座標の入力を回避することができます．このときＮを入力して自分でデータを入力する場合，座標データのほかに，線を引くのか座標を置くだけなのかを与えるパラメータが必要となるため，線を引く最初の点を０で与え，次々に１を与えてゆくことで点を結ぶことができます．

(2) 光線追跡法(ray-tracing)

今まで，パソコンによるコンピュータ・グラフィックス（CG）のほとんどが，ワイヤーフレームによるものでした．しかし，現在のCGと言えば，もうワイヤーフレームのものはあまり見かけず，面画のものばかりです．従来の機種では，ドットごとに8色付けられるだけで，しかもその8色が原色の8色でしたから，これも無理からぬことと言えます．一方SMC-777は解像度は320×200とやや低めながら，ドットごとに16色を付けられますし，オプションのカラーパレットを装着すれば4096色から16色が選べるようになります．これでもコンピュータ・グラフィックスの最先端で使われている機械に比べればまだまだ非力ですが，面画の雰囲気ぐらいは味わえます．ここまでにも，いくつか面画は出てきましたが，ここでは現在のCGで多く使われている光線追跡法による面画を描いてみます．

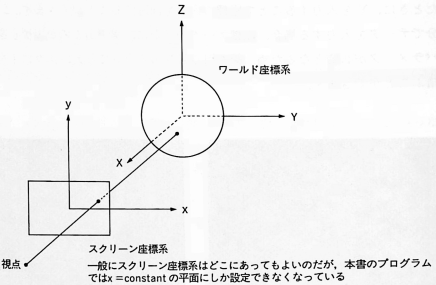


図6-20　光線追跡法の概念

原理

光線追跡法というとワイヤーフレームによるものより，さらに難しい数学的知識が必要と思われる方も多いかもしれません．しかし，単純なものを描く場合はむしろ光線追跡法のほうが簡単ではないか，と筆者は感じています．後述するプログラムに使ってある数学は高校2年までに学習することばかりです．

光線追跡法とは，3次元空間にある物体を，視点を定めて，ある平面（＝画面）に投影する手法です．視点から画面上のある点に向かって直線を引き，それをどんどん延長していきます．その直線が物体とぶつからなければ画面上のその色は背景色ですし，ぶつかったら，ぶつかった点の色が画面上の点の色となります．このとき，物体の位置の様子を記述するためのワールド座標系と，画面に対応する平面であるスクリーン座標系（これは2次元）の2つが必要となります．

球とその影を描く

ここでは，球が床に影をおとしている様子を描きます．一般にスクリーン座標系は任意の場所に置けるのですが，簡単にするために，スクリーン座標系はワールド座標系のYZ平面と平行な平面（X＝constantで記述される平面）のところにしか置けなくなっています．そしてスクリーン座標系の原点はワールド座標系のX軸上に固定されています．またスクリーン座標系と実際の画面とは座標値が同じになっています．たとえば，スクリーン座標系の（10,20）は画面上の（10,20）に対応しているわけです．

カラーパレットの説明の項で描いた球体と同様に，平行光線が球に当たり，球の各点の明るさは，その点に接する平面の法線ベクトルと，光源の方向へ向かうベクトルとのなす角の cos θ に比例するとしています．床のある点が影になるのは，その点から光源方向に引いた直線が球と交わるときです．床の色はカラーコード1で，影になっているところは背景と同じ色（カラーコード0）となります．

数学的に説明すると，まず，

・視点＝（X_{VP}, Y_{VP}, Z_{VP}）
・スクリーン座標系は$X=X_{CS}$の位置にある
・球：$(X-X_{CS})^2+(Y-Y_{CS})^2+(Z-Z_{CS})^2=R^2$
・床：$aX+bY+cZ+d=0 \quad (X_{LS1}<X<X_{LS2}, Y_{LS1}<Y<Y_{LS2})$

とします．すると，視点と画面上の一点（X_{SC}, Y_{SC}, Z_{SC}）を結ぶ直線はtを媒介変数とすると，

$$X=X_{VP}+(X_{SC}-X_{VP})t$$
$$Y=Y_{VP}+(Y_{SC}-Y_{VP})t$$
$$Z=Z_{VP}+(Z_{SC}-Z_{VP})t$$

となります．これを球の式に代入すると，

$$\{X_{VP}-X_{SC}+(X_{SC}-X_{VP})t\}^2$$
$$+\{Y_{VP}-Y_{SC}+(Y_{SC}-Y_{VP})t\}^2$$
$$+\{Z_{VP}-Z_{SC}+(Z_{SC}-Z_{VP})t\}^2=R^2$$

となり，tに関する二次方程式になります．この二次方程式の判別式Dにより，

D＞0：直線が2点で交わる
D＝0：直線が球と接する
D＜0：直線は球とは交わりも接しもしない

と場合分けできます．そして2点で交わるとき，tの2つの解をt_1, t_2（$t_1<t_2$）とすると，直線が交わる点のうち視点に近い点は，

$$(X_{VP}+(X_{SC}-X_{VP})t_1,\ Y_{VP}+(Y_{SC}-Y_{VP})t_1,\ Z_{VP}+(Z_{SC}-Z_{VP})t_1)$$

となります．また直線の式を床の式に代入すると，

$$\begin{aligned} &a(X_{VP}+(X_{SC}-X_{VP})t) \\ &+b(Y_{VP}+(Y_{SC}-Y_{VP})t) \\ &+c(Z_{VP}+(Z_{SC}-Z_{VP})t)+d=0 \end{aligned}$$

となり，tの一次方程式となります．この方程式の解をt_3とすると，これがt_1より大きければ床のほうが遠くにあり，球が見えることになります．

プログラム

リスト6-20にプログラムを示します．1230行以降のDATA文に図形のデータが入っています．これを順に説明すると，

1230：視点の座標
1240：スクリーン座標系がワールド座標系のX＝いくつのところにあるか
1250：床を表す平面の法線ベクトル
1260：床を表す平面が含む点１つ
1270：球の中心
1280：球の半径
1290：光源の方向のϕ，θを度（degree）で表したもの
1300：床は長方形とするが，その対角のうち一方のX座標とY座標
1310：もう一方のX座標とY座標
1320：計算する画面の範囲も長方形であるが，その対角のうち一方
1330：もう一方
1340：ドットの横の長さ／縦の長さ

RUNさせると二次元の座標値を４つ表示しますが，これは視点から画面の四隅に直線を引いたときに，床を含む平面と交わる点のX座標とY座標です．そして，

O.K.?

とたずねてきますが，このときYまたはyを入力すると計算を始めます．そして，これ以外を入力すると，図形のデータの部分のリストを表示してプログラムを終了します．

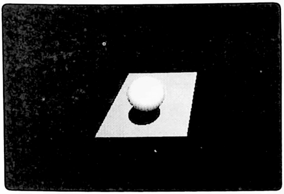
計算中に左端上に数字が表示されていますが，これはそのとき計算している画面上の点のY座標です．ただし，画面の座標系はGLOCATEにより，中心が原点で，右と上がそれぞれX座標とY座標の正方向になっています．

List 6-20

```
10 '
20 '============== draw a sphere and a floor ==============
30 '
40 WIPE
50 '
60 DEF FNRAD(DEG)=DEG*PI/180
70 '
80 READ XVP,YVP,ZVP 'view point
90 '
100 READ XSC        'x of screen
110 '
120 READ XNL,YNL,ZNL 'normal line vector of the floor
130 READ XFC,YFC,ZFC 'a point the floor contains
140 ACF=-(XNL*XFC+YNL*YFC+ZNL*ZFC)
150  'acf : another constant for plain specification
160  'xnl*x+ynl*y+znl*z+acp=0
170 '
180 READ XCS,YCS,ZCS 'center of the sphere
190 READ RS         'radius of the sphere
200 '
210 READ PHI,THE     'phi and theta of the light
220 XLV=COS(FNRAD(PHI))*SIN(FNRAD(THE))
230 YLV=SIN(FNRAD(PHI))*SIN(FNRAD(THE))
240 ZLV=COS(FNRAD(THE))
250  'xlv,ylv,zlv : light vector
260 '
270 READ XLF1,YLF1,XLF2,YLF2 'limits of the floor
280 READ YPL1,ZPL1,YPL2,ZPL2 'limits of the physical screen
290 READ RAT             'ratio of horizontal to vertical
300 '
310 YSC=YPL1*RAT : ZSC=ZPL2 : GOSUB *VIAREA
320 PRINT USING "#####.#_,#####.#";XIS,YIS;
330 YSC=YPL2*RAT : GOSUB *VIAREA
340 PRINT USING "    #####.#_,#####.#";XIS,YIS
350 YSC=YPL1*RAT : ZSC=ZPL1 : GOSUB *VIAREA
360 PRINT USING "#####.#_,#####.#";XIS,YIS;
370 YSC=YPL2*RAT : GOSUB *VIAREA
380 PRINT USING "    #####.#_,#####.#";XIS,YIS
390 PRINT "O.K.?" : I$=INKEY$(1)
400 IF I$<>"Y" AND I$<>"y" THEN LIST *ODATA- : END
410 '
420 CCLEAR
430 CURSOR OFF
440 CPALETTE OFF
450 FOR I=2 TO 15
460   DEF PALETTE I,0,0,I    'color of the sphere
470 NEXT
480 DEF PALETTE 1,4,3,0      'color of the floor
490 GLOCATE(160,100),0,-1
500 '
510 '-- compute --
520 '
530 XVC=XVP-XCS : XSV=XSC-XVP
540 CQ2=XVC*XVC-RS*RS : BQ2=XVC*XSV : AQ2=XSV*XSV
550  'a,b,c of quadratic equation
560 BS2=XNL*XVP+ACF : AS2=XNL*XSV
570  'a,b of simple equation
```

```
580 '
590 FOR ZSC=ZPL2 TO ZPL1 STEP -1
600   PRINT (0,0)USING "###";ZSC
610   ZVC=ZVP-ZCS : ZSV=ZSC-ZVP
620   CQ1=CQ2+ZVC*ZVC : BQ1=BQ2+ZVC*ZSV : AQ1=AQ2+ZSV*ZSV
630   BS1=BS2+ZNL*ZVP : AS1=AS2+ZNL*ZSV
640   '
650   FOR YPSC=YPL1 TO YPL2
660     YSC=YPSC*RAT
670     YVC=YVP-YCS : YSV=YSC-YVP
680     CQE=CQ1+YVC*YVC : BQE=BQ1+YVC*YSV : AQE=AQ1+YSV*YSV
690     BSE=BS1+YNL*YVP : ASE=AS1+YNL*YSV
700     '
710     PTF=-BSE/ASE
720       'ptf : parameter T at intersection with the floor
730     XIS=XVP+XSV*PTF : YIS=YVP+YSV*PTF : ZIS=ZVP+ZSV*PTF
740       'intersection with the floor
750     '
760     DQE=BQE*BQE-AQE*CQE
770     IF DQE<0 THEN GOSUB *SNVISIBLE ELSE GOSUB *SPVISIBLE
780    NEXT
790 NEXT
800 BEEP : BEEP
810 END
820 '
830 '-- sphere not visible --
840 '
850 *SNVISIBLE
860  IF XIS<XLF1 OR XIS>XLF2 THEN RETURN
870  IF YIS<YLF1 OR YIS>YLF2 THEN RETURN
880 '
890  XIC=XIS-XCS : YIC=YIS-YCS : ZIC=ZIS-ZCS
900  CQE=XIC*XIC+YIC*YIC+ZIC*ZIC-RS*RS
910  BQE=XIC*XLV+YIC*YLV+ZIC*ZLV
920  ' XLV*XLV+YLV*YLV+ZLV*ZLV=1
930  IF BQE*BQE-CQE<0 THEN GPLOT(YPSC,ZSC),1
940  RETURN
950 '
960 '-- sphere probably visible --
970 '
980 *SPVISIBLE
990  PTS=(-BQE-SQR(DQE))/AQE
1000  IF XIS<XLF1 OR XIS>XLF2 THEN 1030
1010  IF YIS<YLF1 OR YIS>YLF2 THEN 1030
1020  IF PTS>PTF THEN GOSUB *SNVISIBLE : RETURN
1030  XIS=XVP+XSV*PTS : YIS=YVP+YSV*PTS : ZIS=ZVP+ZSV*PTS
1040    'intersection with the sphere
1050  CLR=((XIS-XCS)*XLV+(YIS-YCS)*YLV+(ZIS-ZCS)*ZLV)/RS
1060  IF CLR<0 THEN CLR=2 ELSE CLR=CLR*13+2
1070  GPLOT(YPSC,ZSC),CLR
1080  RETURN
1090 '
1100 '-- visible area of the plain which contains the floor --
1110 '
1120 *VIAREA
1130  XSV=XSC-XVP : YSV=YSC-YVP : ZSV=ZSC-ZVP
1140  BSE=XNL*XVP+YNL*YVP+ZNL*ZVP+ACF
1150  ASE=XNL*XSV+YNL*YSV+ZNL*ZSV
1160  PTF=-BSE/ASE
```

```
1170  XIS=XVP+XSV*PTF : YIS=YVP+YSV*PTF
1180  RETURN
1190 '
1200 '-- data of the object --
1210 '
1220 *ODATA
1230  DATA  500, 100,   0   :'view point
1240  DATA  250              :'x of screen
1250  DATA    1,   0,   1   :'normal line vector of the floor
1260  DATA  -80,   0, -80   :'a point the floor contains
1270  DATA    0,-100,   0   :'center of the sphere
1280  DATA   40              :'radius of the sphere
1290  DATA   25,  65         :'phi and theta of the light
1300  DATA -233,-250         :'limits of the floor
1310  DATA  -50, -19
1320  DATA        -70, -50  :'limits of the physical screen
1330  DATA         70,  25
1340  DATA  .85
1350     'ratio of horizontal length of the pixel to vertical
```


光線追跡法による球とその影

6-6 SMCグラフィックスエディターの利用

これまでの説明ではBASICにおけるグラフィックスに主眼が置かれていました．計算式で表せる図形の絵を描くときはBASICのコマンドを使うのもなかなか有効なのですが，ただ単に絵を描くときには不便です．そこで，それ専用のツールを使うことを誰しもが考え，現在世の中には「グラフィック・エディター」と称するソフトが数多く出回っています．ソニーの「SMCグラフィックスエディター(SMW-S001D)」もそんななかの1つなのですが，別にソニーにこびるわけではありませんが，グラフィックスをエディットするツールとしてはよくできたものと言えます．本節ではこの「SMCグラフィックスエディター」(仕下GEと呼びます)を紹介し，このエディタとBASICを組み合わせた利用方法を追求していきます．

(1) 使い方

まず起動するにはSony Filerのシステムメニューから GREDIT というものを選択します．これでGEの「エディットモード」に入ります．GEにはこの「エディットモード」と「コマンドモード」とがあり ESC により切り換わります．コマンドの実行やコマンドに関する指示は基本的にファンクションキーで行うようになっています．それぞれのキーの機能と若干の情報が画面に表示されていますが，これはスペースバーを押すことにより，画面の下端に表示するか，画面の上端に表示するか，表示しないかを選択することができます．また H を押すとそのときの状態に応じた解説が表示されます．

エディットモード

最も基本的な，ドットごとのエディットは，エディットモードで行います．画面には十字型のカーソルが表示され，これはカーソルパドルにより縦，横，斜めに1ドットずつ，シフトキーと同時に押すことにより8ドットずつ，移動します．ドットの色を変えるには表6－6に示すような，色に対応するキーを押します．また CTRL と同時にカーソルパドルを押すことにより，カーソルの位置の色を隣にコピーすることができます．

カラーコード	対応するキー	カラーコード	対応するキー
0	K	8	1
1	B	9	2
2	G	10	3
3	T,C	11	4
4	R	12	5
5	P,M	13	6
6	Y	14	7
7	W	15	8

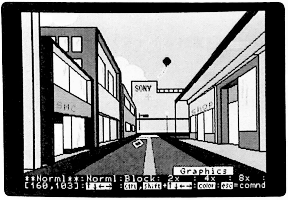
表6-6　カラーコードとキーの対応

エディットモード中にはさらに，

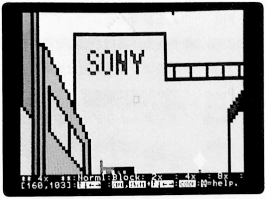
Norml	Block	2×	4×	8×

の5つのモードがあり，Normalモード以外ではカーソルが8画素×8画素の四角(画素はディスプレイ上の1ドットのこと)になり，ドットの変更はこの四角のなかが一度に行われます．

2×，4×，8×はそれぞれ，2倍，4倍，8倍に画面を拡大するモードで，カーソルが8画素×8画素と一定ですから，カーソルに含まれるドット数はそれぞれ16，4，1となります．従ってドット単位の修正がやりやすくなるわけです．Iキーを押すと表示されている拡大された絵を，もとの絵の代わりにNormalモードの絵とすることができます．



SMCグラフィックスエディターで描いた町並み
(Normal モード)



(4×モード)

Block モードは画面はそのままで，カーソルだけが四角に変ります．従って Block モード中で CTRL を押しながらカーソルパドルを押すと，カーソルの部分の 8 画素× 8 画素の範囲が隣の 8 画素× 8 画素の部分に複写されます．

また，CTRL を押しながらカーソルパドルを押したり，単独に Q キーを押したりすると，カーソルの部分のデータがバッファに蓄えられ，Z キーを押すことによりそのデータが読み出されます．

コマンドモード

エディットモードの Normal モードから，ESC を押すことによりコマンドモードに移ります．コマンドモード中で ↑ や ↓ を押すとファンクションキーにより実行されるコマンドが変り，併せて表示も変ります．以下にコマンドを列挙します．

Clear ：画面全体をある色で塗りつぶす
Color ：特定のカラーコードの部分を別のカラーコードに変える
Load ：絵のファイルを読み込む．その際読み込んだデータをどこに表示するかをファンクションキーで指定する．Homeを指定すると画面一杯に表示され，Loadを指定するとカーソルの位置が読み込まれる絵の左端上となる
Ovlay ：絵のファイルを，左端上の色を透明色（6-3参照）として読み込む
Save ：絵のファイルを書き込む．その際長方形の範囲を指定してセーブすることができる
Line ：2 点を結ぶ直線を描く
Box ：対角を指定して箱を描く
FBox ：中を塗りつぶした箱を描く
Circ ：中心と円周の 1 点を指定して円を描く
FCirc ：中を塗りつぶした円を描く

Fill　：ある部分を塗りつぶす
Copy　：長方形の範囲を別のところに複写する
Text　：グラフィック画面に文字を書く
Print　：プリンタに絵を出力する
Bordr　：ボーダーエリアの色を変える
Palet　：パレットの色を変更する
Move　：長方形の範囲を別のところに移す
Edge　：図形のエッジを検出して図形を囲んで線を引く
Scale　：図形の拡大，縮小をする
Delet　：セーブされた絵のファイルを消去する
Renam：セーブされた絵のファイルの名前を変更する
Exit　：システムメニューに戻る

(2) BASICによりGEのファイルを読み込む

GEで作成した絵のファイルは，

〈GEのSaveコマンドで指定したファイル名〉．PIC

というファイル名でディスクに格納されます．このファイルはBASIC中でもリンクモジュールPIC.PACに入っている拡張コマンドにより読み込めますし，BASIC上で描いたグラフィックスをGEの作るファイルと同じ形式でディスクにセーブすることができます．まずロードには，

_GLOAD(〈ファイル指定〉[,〈左上端のX座標〉,〈左上端のY座標〉])

という拡張コマンドを用います．〈ファイル指定〉に拡張子は必要ありませんし，もし仮に拡張子を付けたとしても，それとは無関係に.PICが拡張子として使われます．また座標値を省略すると，座標値としては，画面の左上端が採用されます．この_GLOADはBASICのGPUTで透明色を指定しなかったときと，同様な動作をします．透明色を指定したいときは，

_GOVLY(〈ファイル指定〉[,〈左上端のX座標〉,〈左上端のY座標〉,〈透明色のカラーコード〉])

を使います．〈透明色のカラーコード〉を省略したときは絵のデータの左上端の色が透明色として採用されます．

次に，グラフィック画面を全部セーブしたいときは，

_GSAVEALL(〈ファイル指定〉,〈ボーダーエリアのカラーコード〉)

を使い，画面の一部をセーブしたいときは，

__GSAVE (〈ファイル指定〉[，〈始点座標X〉，〈始点座標Y〉，〈X方向のドット数〉，〈Y方向のドット数〉])

とします．なお〈始点座標X〉と〈X方向のドット数〉は偶数で指定しなければなりません．

では，GEを用いて作るゲームプログラムを紹介してみましょう．まず，図6-21に示すような画面を作り，ディスクに〝MOGURA〞と名付けてセーブします．次にリスト6-21に示すプログラムを作り，ゲーム(モグラたたき)にするわけですが，ここで注意しなければならないのは，GE中でパレットの色を変更しても，GLOADで読み込むときには，そのパレットの内容は読み込まれないことです．そのために，GE中で指定したとおりに，DEF PALETTEを用いて指定しなければなりません．リスト6-21を見てもわかるように，このプログラムではDEF PALETTE以外のグラフィックコマンドは一切使っていませんが，それでもこのようなゲームを作ることができるのです．

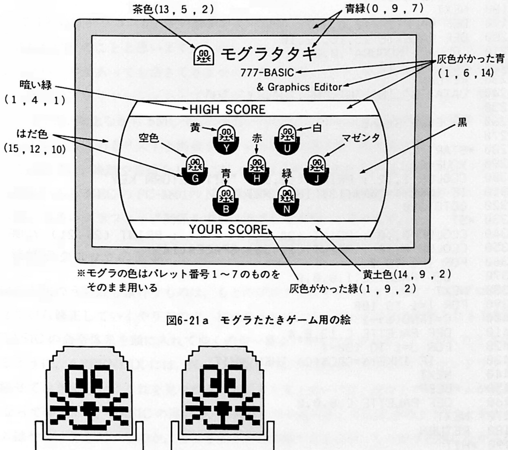


図6-21a モグラたたきゲーム用の絵

図6-21b モグラ拡大図

このようにSMCグラフィックスエディターは単体としての完成度もさることながら，単なる「お絵描きツール」に留まらず，作成したデータを汎用的に扱えるようになっている，トータルな完成度の高いツールと言えます．

List 6-21

```
 10 '
 20 '-- MOGURA TATAKI GAME --
 30 '
 40  GOSUB *INIT
 50  GOSUB *START
 60  GOSUB *GMOV
 70  GOTO 50
 80 '
 90 '-- Initialize routine
100 '
110 *INIT
120  WIPE : CONSOLE 40 : CURSOR OFF : CPALETTE OFF
130  DEF INT A-Z
140  DIM CRC$(7)
150  RESTORE 240
160  FOR I=1 TO 7
170    READ CRC$(I)
180  NEXT
190  DEF PALETTE 12,0,9,7
200  DEF PALETTE 15,15,12,10
210  _GLOAD("MOGURA",0,0)
220  GCOLOR ,,12
230  RETURN
240  DATA "B","N","G","H","J","Y","U"
250 '
260 '-- Game Start
270 '
280 *START
290  YOUR=0
300  CCOLOR 1,,2 : PRINT (12,23) "HIT RETURN KEY"
310  IF INKEY$=CHR$(13) THEN *ST
320  GOTO 310
330 *ST
340  CCOLOR 0,,0 : PRINT (24,21) "    " : PRINT (24,21) YOUR
350  CCOLOR 0,,0 : PRINT (12,23) SPACE$(14)
360  FOR I=1 TO 7
370    DEF PALETTE I,0,0,0
380  NEXT
390  FOR I=1 TO 100
400    C=IRND(6)+1
410    DEF PALETTE C,13,8,4
420    FOR J=1 TO 130-I
430      IF INKEY$=CRC$(C) THEN *HIT
440    NEXT
450    BEEP
460    DEF PALETTE C,0,0,0
470  NEXT
480  RETURN
490 *HIT
500  DEF PALETTE C,15,0,0
```

```
510   _PANIC(50)
520   YOUR=YOUR+1+I¥10
530   PRINT (24,21) YOUR
540   GOTO 460
550  ′
560  ′-- Game Over
570  ′
580  *GMOV
590   FOR I=1 TO 7
600     DEF PALETTE I,13,8,4
610   NEXT
620   CCOLOR 4,,2 : PRINT (13,23) 'GAME OVER !!'
630   CCOLOR 1,,0
640   IF YOUR>HIGH THEN HIGH=YOUR : PRINT (24,7) HIGH
650   PAUSE 100
660   RETURN
```

6-7 N_{88}-BASICからの移植テクニック

SMC-777というパソコンが，使い勝手がよいとか，グラフィックスが美しいとかいうことは，もう十分おわかりのことと思います．しかし，コンピュータはソフトが充実していなくては，どんなによいハードがあっても活きてきません．そこで多くのパソコンユーザーは雑誌や書籍に載っている様々なアプリケーション・プログラムのリストを自分のマシンに打ち込んで楽しむことになるのですが，残念ながらSMC-777の場合(今のところ!)，それほどメジャーな機種でないために，そのまま打ち込めるリストと出会うチャンスはそれほど多くありません．

そこで本節では，雑誌や書籍などでしばしばリストの掲載される(つまりメジャーな！)パソコンの代表機種であるNECのPC-8801のN_{88}-BASICから，777 BASICへの移植テクニックを，機種間の違いの多いグラフィック関係を中心に追求してみたいと思います．

(1) 移植テクニックの基本

移植の方法のうち最も素朴なものは，もとのプログラムをそのまま打ち込んで，プログラムを走らせながら修正していくやり方です．初めのうちは，そうやって移植の対象となるBASICと777 BASICの命令体系を頭に入れてゆくのが一番よいかもしれません．こういったときの助けとなるように，APPENDIXには，N_{88}-BASICから777 BASICのクイック·コマンドリファレンスを載せておきました．これを見ながらコマンドを覚えていけば，段々とプログラムが見えるようになってゆきます．BASICの違いや，共通点をのみ込みさえすれば，どのようなことをこのプログラムはやろうとしているのか，どこを変えれば移植できるのか，しっかり把握しながら作業を進めることができます．

777 BASICで何かわからないことがあれば，この際しっかりマニュアルを見て覚えてしまい

ましょう．自由自在に操ることができるようになれば，移植なんて簡単なものです．そうなれば移植のたびに，別の機種のBASICを見る目が開かれていくのに気が付くでしょう．

さて，グラフィック関係のプログラムの移植では，次の3つに関係する部分が移植の鍵になります．

・画面の初期設定コマンドの違い
・解像度の違い
・画素の違い(色の数)

これらの違いを押さえてしまえば，あとはコマンドの細かな差異を修正するだけになります．そこでN_{88}-BASICのこれらに関係する事柄について詳しく説明していきましょう．

画面の初期設定

N_{88}-BASICでグラフィック画面のモードを設定するコマンドはSCREENです．

```
SCREEN　a，b，c，d
```

という型で表せるこのコマンドですが，aには0，1，2の値が入り，それぞれの意味は，

0 ：カラーモード(640×200ドット)
1 ：白黒モード(640×200ドット)
2 ：高分解能白黒モード(640×400ドット)

となります．b，c，dは，SMC-777にはない機能，たとえば表示するページを切り換えるときなどに使われる指定なので，移植の際には無視します．

テキスト画面のモードを設定するコマンドはCONSOLEで，

```
CONSOLE a，b，c，d
```

という型になっています．aは文字表示開始行を，bは文字表示行数を指定していますが，cはファンクションキーの内容表示のON／OFFを表していて，SMC-777では関係ありません．dに入る値は，テキスト画面の色に関するもので，

0 ：テキスト画面を白黒モードにする
1 ：テキスト画面をカラーモードにする

ということを指定します．

各部の色を指定するコマンドとしては，COLORがあり，

```
COLOR a，b，c，d
```

という型．aに入る値は文字の色，bに入る値はバックグラウンドカラー，cに入る値はボーダーカラー，dに入る値はグラフィックの表示カラーを指定します．なお，N_{88}-BASICにおけるカラーコードは，

0 ― 黒　1 ― 青　2 ― 赤　3 ― 紫
4 ― 緑　5 ― 水色　6 ― 黄色　7 ― 白

の8つとなっています．

画面のクリアに関係するコマンドはひとつだけで，

```
CLS a
```

という型をしており，aの値により，

1 ：文字だけを消す
2 ：グラフィックスだけを消す
3 ：両方とも消す

という指定になります．

解像度と色

SMC-777の16色モードの解像度は320×200ドット．これに対してPC-8801はカラーで640×200ドットの解像度がありますので，何も考えずに移植しようとすると，絵がはみ出してしまいます．また，もっとドットの粗い解像度に対して書かれたプログラムをそのまま移植すれば，小さすぎて見栄えがしないものになってしまいます．適度な大きさに移植する方法はいろいろ考えられますが，これについてはこのあとのサンプル・プログラムを参考にしてください．

色の問題については，SMC-777はカラーパレットを装着すれば，4096色から16色を選べるので，該当する色がないという心配はありません．白黒や8色のプログラムも工夫して，移植後の方が美しいと言われるように頑張りましょう．

（2）サンプル・プログラム

DRAGON CURVE

さてプログラムです．このプログリム（リスト6-22）はN_{88}-BASICで書かれており，リスト6-23は777 BASICで書かれています．これはグラフィックのきれいな，その独特の動き方から「竜」の名前を付けられた数学的曲線を組み合わせたデモンストレーションです．

List 6-22

```
1000 '
1010 '-- DRAGON CURVE --
1020 '
1030  SCREEN 0,0,0,1 : CONSOLE 0,25,0,1 : COLOR 7,0,0,7 : CLS 3
1040  DEFINT A-Z
1050  LONG=1 : DIR=2 : POINT(320,100) : GOSUB *DIRSET
1060 '
1070  FOR D=1 TO 18
1080  LOCATE 0,0 : PRINT "DEAPTH"D
1090  COLOR ,,,1+(D-INT(D/7)*7)
1100  GOSUB *PROCL
1110  NEXT : END
1120 *PROCL
1130  IF D=0 THEN LINE -STEP(LONG*X,LONG*Y) : RETURN
1140    D=D-1
1150    GOSUB *PROCL
1160    GOSUB *LEFT
1170    GOSUB *PROCJ
1180    D=D+1
1190  RETURN
1200 *PROCJ
1210  IF D=0 THEN LINE -STEP(LONG*X,LONG*Y) : RETURN
1220    D=D-1
1230    GOSUB *PROCL
1240    GOSUB *RIGHT
1250    GOSUB *PROCJ
1260    D=D+1
1270  RETURN
1280 *RIGHT
1290  DIR=DIR+1 : IF DIR>4 THEN DIR=1
1300  GOTO *DIRSET
1310 *LEFT
1320  DIR=DIR-1 : IF DIR<1 THEN DIR=4
1330  GOTO *DIRSET
1340 *DIRSET
1350  ON DIR GOTO 1360,1370,1380,1390
1360  X= 0 : Y=-1 : RETURN
1370  X= 1 : Y= 0 : RETURN
1380  X= 1 : Y= 1 : RETURN
1390  X=-1 : Y= 0 : RETURN
```

List 6-23

```
1000 '
1010 '-- DRAGON CURVE --
1020 '
1030  GMODE 1 : GCOLOR 7,0,0,0 : GCLEAR : CCLEAR : CURSOR OFF
1040  DEF INT A-Z
1050  LONG=1 : DD=2 : X1=160 : Y1=100 : GOSUB *DIRSET
1060 '
1070  FOR D=1 TO 18
1080    LOCATE (0,0) : PRINT "DEAPTH";D
1090    GCOLOR 1+(D-INT(D/7)*7)
1100    GOSUB *PROCL
1110  NEXT : END
```

```
1120 *PROCL
1130  IF D=0 THEN *FORWARD
1140    D=D-1
1150    GOSUB *PROCL
1160    GOSUB *LEFT
1170    GOSUB *PROCJ
1180    D=D+1
1190  RETURN
1200 *PROCJ
1210  IF D=0 THEN *FORWARD
1220    D=D-1
1230    GOSUB *PROCL
1240    GOSUB *RIGHT
1250    GOSUB *PROCJ
1260    D=D+1
1270  RETURN
1280 *RIGHT
1290  DD=DD+1 : IF DD>4 THEN DD=1
1300  GOTO *DIRSET
1310 *LEFT
1320  DD=DD-1 : IF DD<1 THEN DD=4
1330  GOTO *DIRSET
1340 *DIRSET
1350  ON DD GOTO 1360,1370,1380,1390
1360  X= 0 : Y=-1 : RETURN
1370  X= 1 : Y= 0 : RETURN
1380  X= 0 : Y= 1 : RETURN
1390  X=-1 : Y= 0 : RETURN
1400 *FORWARD
1410  X2=X1+LONG*X : Y2=Y1+LONG*Y
1420  LINE (X1,Y1)-(X2,Y2)
1430  X1=X2 : Y1=Y2 : RETURN
```

このプログラムでは，LINE 文の変更が主になります．と言っても最初から難関が待ち受けています．スクリーンの初期設定が，それです．N_{88}-BASIC(1030行)のSCREEN 0,0,0,1は，カラーモード指定で解像度640×200であることを示しています．続いてCONSOLE 0,25,0,1：COLOR 7,0,0,7：CLS 3で画面をクリアしていますが，777 BASICではこれをGCOLOR 7,0,0,0以下の命令で行っています(1030行)．よく777 BASICのマニュアルを見て，何をしているのか理解してください．この1行は，指定する内容は変っても，大部分のプログラムの先頭付近に存在する初期設定の部分です．

さてCURSOR OFFとは何でしょう？　SMC-777を使っていればわかることですが，プログラムの実行中には不必要なカーソルを消す命令なのです．

本題のLINE文に移ります．777 BASICのLINE命令は，ごく一般的なものですが，STEPを使った相対指定ができません．つまり，LINE(X, Y)－(X+α, Y+α) とは書けても，LINE (X, Y)－STEP(α, α)と書くことができません．そこでプログラムによって，

X′=X+α : Y′=Y+α

LINE(X, Y)−(X′, Y′)

X=X′ :Y=Y′

を実行しなければなりません．もうひとつ，LINE 文と GPLOT 文を使って，

GPLOT(X, Y)

LINE−(X+α, Y+α)

を実行すると，LINE(X, Y)−(X+α, Y+α)と同等の効果があります．これは GPLOT 文で指定された点や，他のグラフィック命令で，最後に指定された点を BASIC は覚えていて，LINE−(X+α, X+α)が実行されるとき，最初の点を補うからです．

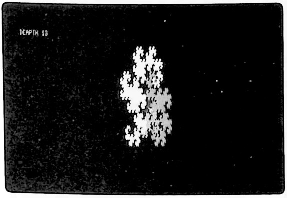
N_{88}-BASIC で言えば，GPLOT は PSET であり，全く同じことをしますが，N_{88}-BASIC は POINT という命令があり，これは，BASIC にその点を覚えてもらうだけの働きをします．これがリスト 6-22の1050行のPOINTの正体です．

以上のことを頭に入れて，もう一度リスト 6-22を見てください．少しは展望がよくなったのではないでしょうか．なお DRAW 文を使った移植の一例としてリスト 6-24を挙げておきますので参考にしてください．

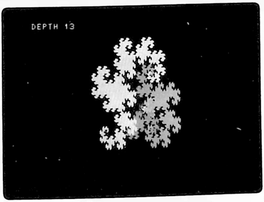
List 6-24

```
1000 '
1010 '-- DRAGON (DRAW VERSION) --
1020 '
1030  GMODE 1 : GCOLOR 7,0,0,0 : GCLEAR : CCLEAR : CURSOR OFF
1040  DEF INT A-Z
1050  DRAW "S1Z0m160,100" : DD=2
1060 '
1070  FOR D=1 TO 18
1080    LOCATE (0,0) : PRINT "DEAPTH";D
1090    C=1+(D-INT(D/7)*7) : DRAW "C(C)"
1100    GOSUB *PROCL
1110  NEXT : END
1120 *PROCL
1130  IF D=0 THEN DRAW "R" : RETURN
1140     D=D-1
1150     GOSUB *PROCL
1160     GOSUB *LEFT
1170     GOSUB *PROCJ
1180     D=D+1
1190   RETURN
1200 *PROCJ
1210  IF D=0 THEN DRAW "R" : RETURN
1220    D=D-1
1230    GOSUB *PROCL
1240    GOSUB *RIGHT
1250    GOSUB *PROCJ
1260    D=D+1
1270  RETURN
```

```
1280 *RIGHT
1290  DD=DD+1 : IF DD>4 THEN DD=1
1300  GOTO *DIRSET
1310 *LEFT
1320  DD=DD-1 : IF DD<1 THEN DD=4
1330  GOTO *DIRSET
1340 *DIRSET
1350  ON DD GOTO 1360,1370,1380,1390
1360  DRAW "A90"  : RETURN
1370  DRAW "A0"   : RETURN
1380  DRAW "A270" : RETURN
1390  DRAW "A180" : RETURN
```



N_{88}-BASIC によるドラゴンカーブ



777 BASIC によるドラゴンカーブ

基本的には777 BASICへの移植は，パラメータの違い等に気を付ければ比較的ストレートに行えます．あなたのSMC-777にもドラゴンを走らせてください．

3-D SHEET

ダイナミックな動きのドラゴンのあとには，静かな美しいプログラムを移植してみましょう．ワイヤーフレームのコンピュータ・グラフィックスに，パレット処理を使った回転する球体を乗せています．

リストは2つあり．それぞれ N_{88}-BASIC と777 BASICで記述されています．

さて N_{88}-BASIC の640×200，8色のグラフィックを移植する際777 BASICでも8色を使うことにして320×200の画面を採用すると，X軸が半分になってしまいます．そこでX軸に関する設定を半分にして対応するわけですが(リスト6-23,1050行参照)，このプログラムでは計算が複雑で，そういった処理が困難です．そこで，入口がだめなら出口をということで，GPLOTやLINEなどの出力部で値を半分にしてしまっています．

1180，1240，1260，1280行を見てください．多少効率は落ちますが，このように複雑なプログラムで，結果の比率だけが問題なときには，作業効率からいっても，この方法は有効な手段とい

えるでしょう．ただし出力部の数が多いときや，プログラムの流れが単純なものは，設定をやり直した方が美しくなります．

1330～1370行ではCIRCLEの半径を，777ではX軸とY軸に分けて記述するための処理です．

1400行以降はパレット機能の違いです．777のパレットはRGBアナログ表示によって16×16×16＝4096色の指定ができるため，N_{88}-BASICのようにRGB 2×2×2＝8色の表示に合わせるには原色を使います．

777ではRGB各色は16段階の色調を持ちますが，赤ならR＝15，G＝0，B＝0，緑ならR＝0，G＝15，B＝0，青ならR＝0，G＝0，B＝15のように8色表示のときには，15と0の組合せで色を決めればよいわけです．

1460行以降では，なるべく単純になるように，もとのプログラムを生かして，3個の球の内で1個だけを異なる色にするという主旨を実現するにとどめています．回転しているように見えるには，最低1個が識別される必要があります．ここではパレットを変えることで物体の移動を実現する，ということを読みとってプログラムを移植したことになります．

移植途上で走らせるうちに，数々のエラー(Syntax ErrorやIlligal function callなど)が出ると思いますが，クイック・コマンドリファレンスを活用して切りぬけてください．筆者が悩んだものとしては，pset命令に対して「配列が範囲を超えています」と言ってくる，というエラーがありました．これはPSETが777 BASICの予約語でないので，PSET(X, Y)を配列だと思われてしまったわけです．

List 6-25

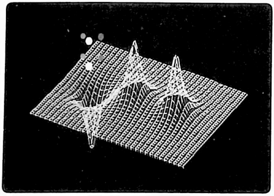
```
1000 '
1010 '-- 3D sheet --
1020 '
1030  DIM MAX(640),MIN(640),BEFORE(85,1)
1040  P=0 : SCREEN 0,0 : CLS 3
1050  FOR I=0 TO 7 : COLOR=(I,I) : NEXT
1060 '
1070  FOR I=1 TO 640
1080    MAX(I)=0 : MIN(I)=200
1090  NEXT
1100  FOR I=1 TO 85 STEP 2
1110    BEFORE(I,0)=2*I+394
1120    BEFORE(I,1)=199-I
1130  NEXT
1140 '
1150  FOR Y=2 TO 130 STEP 2
1160    C=C+1
1170    COLOR ,,,C-INT(C/2)*2+4
1180    POINT(4-3*Y+400,180-Y/2)
1190    FOR X=1 TO 85 STEP 2
1200      S=(X-42)*(X-42) : T=(Y-35)*(Y-35)
1210      U=(Y-70)*(Y-70) : V=(Y-105)*(Y-105)
1220      Z=4500/(S+T+30)+4500/(S+U+30)-4500/(S+V+30)
1230      PY=Z/2+X+Y/2 : PX=2*X-3*Y+400
1240      LINE -(PX,180-PY)
```

```
1250      IF Y=2 THEN *SKIP
1260      LINE(BEFORE(X,0),BEFORE(X,1))-(PX,180-PY)
1270     *SKIP
1280      POINT(PX,180-PY)
1290      BEFORE(X,0)=PX : BEFORE(X,1)=180-PY
1300    NEXT
1310  NEXT
1320
1330  FOR I=1 TO 2
1340    CIRCLE(145,24*I),6,1 : PAINT(145,24*I),1,1
1350    CIRCLE(190,25*I),7,2 : PAINT(190,25*I),2,2
1360    CIRCLE(160,30*I),8,3 : PAINT(160,30*I),3,3
1370  NEXT
1380  C(1)=1 : C(2)=6 : C(3)=1
1390 '
1400  WHILE 1
1410    GOSUB *TURN
1420    COLOR=(4,2) : COLOR=(5,3)
1430    GOSUB *TURN
1440    COLOR=(4,4) : COLOR=(5,5)
1450  WEND
1460 *TURN
1470  FOR I=0 TO 10
1480    FOR J=0 TO 3 : C(J)=C(J+1) : NEXT
1490    C(3)=C(0)
1500    FOR J=1 TO 3
1510      COLOR=(J,C(J))
1520    NEXT
1530  FOR K=1 TO  500 : NEXT
1540  NEXT
1550  RETURN
```

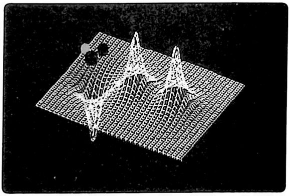
List 6-26

```
1000 '
1010 '-- 3D sheet --
1020 '
1030  DIM MAX(640),MIN(640),BEFORE(85,1)
1040  P=0 : GMODE 1 : GCLEAR : CCLEAR : CURSOR OFF
1050  PALETTE RESET
1060 '
1070  FOR I=1 TO 640
1080    MAX(I)=0 : MIN(I)=200
1090  NEXT
1100  FOR I=1 TO 85 STEP 2
1110    BEFORE(I,0)=2*I+394
1120    BEFORE(I,1)=199-I
1130  NEXT
1140 '
1150  FOR Y=2 TO 130 STEP 2
1160    C=C+1
1170    GCOLOR C-INT(C/2)*2+4
1180    GPLOT((4-3*Y+400)/2,180-Y/2)
1190    FOR X=1 TO 85 STEP 2
1200      S=(X-42)*(X-42) : T=(Y-35)*(Y-35)
1210      U=(Y-70)*(Y-70) : V=(Y-105)*(Y-105)
```

```
1220       Z=4500/(S+T+30)+4500/(S+U+30)-4500/(S+V+30)
1230       PY=Z/2+X+Y/2 : PX=2*X-3*Y+400
1240       LINE -(PX/2,180-PY)
1250       IF Y=2 THEN *SKIP
1260       LINE(BEFORE(X,0)/2,BEFORE(X,1))-(PX/2,180-PY)
1270      *SKIP
1280       GPLOT(PX/2,180-PY)
1290       BEFORE(X,0)=PX : BEFORE(X,1)=180-PY
1300     NEXT
1310  NEXT
1320  '
1330  FOR I=1 TO 2
1340    CIRCLE(72,24*I),6,6,1 : PAINT(72,24*I),1,1
1350    CIRCLE(95,25*I),7,7,2 : PAINT(95,25*I),2,2
1360    CIRCLE(80,30*I),8,8,3 : PAINT(80,30*I),3,3
1370  NEXT
1380  C(1)=1 : C(2)=6 : C(3)=1
1390  '
1400  WHILE 1
1410    GOSUB *TURN
1420    DEF PALETTE 4,0,15,0 : DEF PALETTE 5,0,15,15
1430    GOSUB *TURN
1440    DEF PALETTE 4,15,0,0 : DEF PALETTE 5,15,0,15
1450  WEND
1460 *TURN
1470  FOR I=0 TO 10
1480    FOR J=0 TO 3 : C(J)=C(J+1) : NEXT
1490    C(3)=C(0)
1500    FOR J=1 TO 3
1510      DEF PALETTE J,C(J),C(J),C(J)
1520    NEXT
1530  FOR K=0 TO 500:NEXT
1540  NEXT
1550  RETURN
```



N_{88}-BASIC による3-D SHEET



777 BASIC による3-D SHEET

7章　ハードウェア

SMC-777という機種が，コンパクトなボディに多彩な機能を盛り込んだマシンであるということは，これまでのソフトウェア中心の解説で，大方おわかりいただけたものと思います．コストパフォーマンスの高いパソコンが，このSMC-777といえるわけですが，さらにこの機種の能力を引き出してやろうと思ったなら，どうしてもハードウェアの特性を，より詳しく知らねばなりません．特に機械語のプログラムを作る場合にこのことは顕著になります．本章はこのようなハードに関する若干突っ込んだ解説，さらにハードの特性を利用して，少々毛色の変ったこともできるという可能性の提示をしてみたいと思います．

7-1　ハードウェア構成

(1) 概要

SMC-777はCPUに現在のところ世界的にメジャーなZ-80Aを採用しています．クロックは4.028MHzで，このCPUのクロックも含めてタイミング信号はCX-20120というソニーのカスタムICが作り出しています．メインメモリは64K DRAM 8個で64Kバイト，バンク切り換えをして使うROM（シャドウROMと呼ぶ）が128KのROM 1個で16Kバイト．

キャラクタ表示は80桁×25行と40桁×25行の2つのモードがあり，40桁のときは2画面持つことができます．キャラクタ関係では，キャラクタRAM(CRAM)，アトリビュートRAM(ARAM)，PCGがそれぞれ2Kバイトで合計6Kバイト，そしてCRTCとしてはHD46505S-1が使われています．グラフィックスは320×200でドットごとに16色の指定が可能なモードと，640×200で4色のモードがあります．ビデオRAMは64K SRAM 4個（ちょっとぜいたく）で32Kバイト．これら画面関係のRAMはすべてI/Oにマッピングされています．そしてCX-20121というカスタムICが最終的にビデオ信号を作っています．また，オプションのカラーパレットを装着すれば一気に4096（16^3）色中から16色を選んで使えるようになります．

SMC-777のマイクロフロッピーディスクは256バイト/セクタ，16セクタ/トラック，70トラッ

ク／サーフェスで，片面ながら合計280Kバイトの容量があります．データ転送速度は500Kビット／秒で5インチディスクの2倍，8インチディスクと同じになっています．FDCにはMB8877Aを採用し，データ転送にはDMAを使わずZ-80Aが直接行っています．

サウンド機能としては，PSG SN76489ANにより3重和音とノイズが発生できるほか，CPUが直接ポートのあるビットに0または1を出力することにより，音を発生させることもできます．BASICのPLAYコマンドではCPUにより，SOUNDコマンドとEFFECT.PACではPSGにより，音が発生させられています．

そのほか，キーボードのスキャンは1チップCPU 8041Aがソフトスキャンで行ない，キーが押されたとき割込みをかけることができるようになっています（BASICでは使っていない）．

（2）シャドウROM

SMC-777にはメインメモリ64KとシャドウROM 16Kをバンク切り換えにより使っています．シャドウROMは0000H～3FFFHにマッピングされていて，システムがリセットされたときはシャドウROMが選択されて，このなかのIPLによりシステムが読み込まれます．シャドウROMにはこのほかに，I／O，演算，グラフィックス関係のルーチンやPCGに書き込むフォントデータも入っています．

シャドウROMにバンク切り換えをする手順は一般のバンク切換えより若干賢くなっていますので，以下に説明します．SMC-777では1CHまたは1DHポートに00Hを出力すればROMが，10Hを出力すればRAMが選択されます．しかし，一般のバンク切換えのようにOUT命令の実行直後にバンクが切り換わってしまうのでは，このOUT命令は，バンク切換えをしても変ってしまわない部分（4000H～FFFFH）になければなりません．そこでSMC-777はOUT命令の次の命令が実行された後に切り換わるようになっています．ですから，0000H～3FFFHに，

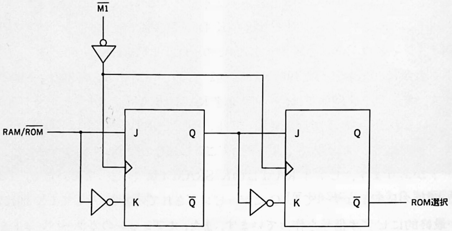


図7-1　シャドウROM選択のメカニズム

```
LD      A, 0
OUT     (1CH), A
JP      0000H
```

という部分があっても，これが実行されれば，しっかりシャドウROMの0000番地へジャンプします．これは，図7-1のようにM1を数えることにより実現されています．この回路を見るとわかるように，さきほど「OUT命令の次の命令が実行されたあと」と書きましたが，これはM1サイクルが2つある命令に関しては成り立たないことになります．ですから，上の例のJP 0000Hの代わりにJP (IX)などは使えないことになります．

(3) 画面

画面構成

6章でも触れたように，SMC-777ではキャラクタ表示とグラフィックス表示とには独立したVRAMがあり，グラフィックスの上にキャラクタが重ねて表示される形になります．また，キャラクタやグラフィックスが表示される部分の外側はボーダーエリアと呼ばれ，この部分は色だけが16色中から指定できます．

画面表示関係のメモリはすべてI/Oにマッピングされていると書きましたが，これにはCレジスタによるIN，OUT命令では，アドレスの上位8ビットにBレジスタの内容が出力されてしまうことを利用しています．そして，画面表示関係のメモリのアクセスにはBレジスタで下位8ビットを，Cレジスタで上位8ビットを指定するようにしています(図7-2)．こうしないと，Cレジスタを使わない通常のIN，OUT命令が使えなくなってしまうからです．

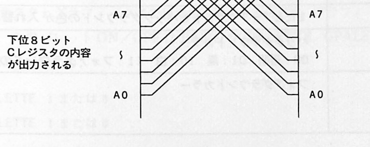


図7-2 VRAMのつながれ方

キャラクタ表示

すでに何度か説明したように，キャラクタ表示に関するメモリにはキャラクタRAM（CRAM），アトリビュートRAM（ARAM），PCG（キャラクタジェネレータがRAMになっていて変更可能）があります．

CRAMには表示する文字のキャラクタコードが入っていて，0000H～07FFHに割り当てられています．領域は2048バイトとられていますが，実際に使用されるのは0000H～07CFHの2000バイトです．そして，40桁モードのときは1番地おきに使用されます．

ARAMはCRAMのそれぞれの番地のキャラクタをどのように表示するかを指定するもので，0800H～0FFFHに割り当てられています（使用されるのは0800H～0FCFH）．指定する内容は，文字の色，文字のバックグラウンドの色，ブリンクさせるか否か，等ですが，ARAMに書き込まれたデータの詳細な意味は表7－1に示します．

Logoのテキストスクリーン状態ではグラフィックスが実際に消されたのではなく，ただマスクされただけになっていますが，これはキャラクタのバックグラウンドカラーを黒（スプリットスクリーン状態やBASICの通常の状態では，透明になっている）にすることにより実現しています．またテキストスクリーン状態からフルスクリーン状態にして，再びテキストスクリーン状態に戻したとき，画面はもとのままになっていて，キャラクタ表示がマスクされたかのように見えます．これは，Logoでは40桁モードしか使わないため通常のキャラクタ表示には偶数番地のCRAM，ARAMを使い，奇数番地のCRAMはスペース，奇数番地のARAMは07Hばかりにしておくのです．こうなっていれば，フルスクリーン状態のときはCRAM，ARAMの奇数番地を表示することによって，キャラクタ表示がマスクされたかのように見せかけることができます．

1つのキャラクタは8×8ドットで構成されていて，キャラクタジェネレータには1文字に対して8バイト必要です．PCGは1000H～17FFHに割り当てられていて，リセットされたときやBASICのFONT RESETが実行されたときには，シャドウROMからフォントデータが読み込まれます．

ビットNo.	内　容
7（MSB）	1だとビット6～ビット0が0000111となったことと同じになる
6	1だとフォアグランドの部分が点滅する
5	1だとフォアグランドとバックグラウンドの色が入れ替わる
4 3	バックグラウンドカラー 00：透明　01：黒　10：白　11：フォアグラウンドの補色
2 1 0（LSB）	フォアグラウンドカラー

表7-1　ARAMに書き込まれるアトリビュート

グラフィック表示

グラフィックのビデオRAM(GRAM)は，8000H～FFFFHに割り当てられています．320×200のときは1バイトに2ドット，640×200のときは1バイトに4ドット割り当てられます．図7-3に画面上のドットとGRAMの対応を示しますが，これを見てもわかるように，ある行の右端とその次の行の左端とは対応するGRAMのアドレスが連続していません．GRAMの割り当てられ方が不連続なのです．BASICのWIPE，GCLEARなどでは連続してGRAMをアクセスしているため(このほうがプログラムも簡単)，APPLEなどと同じようにすだれのような消え方をするのです．

1ドット（横） / 1ドット（縦）

80	00	80	01	80	02		80	9F
90	00	90	01	90	02		90	9F
A0	00	A0	01	A0	02		A0	9F
B0	00	B0	01	B0	02		B0	9F
C0	00	C0	01	C0	02		C0	9F
D0	00	D0	01	D0	02		D0	9F
E0	00	E0	01	E0	02		E0	9F
F0	00	F0	01	F0	02		F0	9F
80	A0	80	A1	80	A2		81	3F
90	A0	90	A1	90	A2		91	3F
FF	00	FF	01	FF	02		FF	9F

図7-3 標準モード(320×200)時の画面のドットとGRAMのアドレスの対応

カラーパレット

6章でも触れましたが，SMC-777では通常カラージェネレータによって16色の表示をしていますが，カラーパレットを取りつけると，4096色から16色選んで使えるようになります．表示に際してカラージェネレータを使うかパレットを使うかはI/Oポートにより選択できますが，この選択はキャラクタとグラフィックスで独立に設定できるのです．グラフィックスのところでも述べましたが，このパレットON／OFFはBASICのCPALETTE，GPALETTEコマンドでできますし，実はLogoにも，

```
CPALETTE 1 または 0
GPALETTE 1 または 0
```

というプリミティブがあり，パレットのON／OFFができます．

(4) 割込み

SMC-777ではモード1の割込み（割込みがかかったら単に38H番地がコールされる）を使います．割込み源としては，60Hz割込み（1／60秒ごとにかかる割込みで，BASICのSOUND命令はこれを使っている）と，キーボードが押されたときの割込み，があります．

60Hz割り込みは，21HポートのLSB（最下位ビット）に1／0を書き込むことより，イネーブル／マスクされます．

キーボードが押されたときの割込みでは1BHポートのLSBに1または0を書き込むことで，割り込みがかかるか否かを選択します．出力された割り込み要求信号は，CPUがキーボードデータを読み込むと，その80μ秒後にリセットされます．もっと速くリセットしたい場合には，1BHポートのMSB（最上位ビット）に1を書き込めば，その約30μ秒後にリセットされます．また，1BHポートのビット4は特別な意味がありますから，書き込むときは0にしておいてください．

(5) SMC-70との違い

SMC-777はハード的にもSMC-70とほとんどコンパチブルになっています．SMC-70のCP／Mのディスクを差し込めばそのままSMC-777でもCP／Mが走るほどです．しかし，変っている点もいくつかあり，その主なものは以下のとおりです．

・グラフィックスの160×100, 640×400のモードがなくなった
・シャドウROMのソケットにSMC-70では32Kバイトのものも差せたが，SMC-777では16Kのものしか差せない
・カレンダークロック，ライトペン・インターフェイスがなくなった

(6) その他

ビットNo.	内　　容
7 (MSB)	[CTRL]が押されていると1
6	[SHIFT]が押されていると1
5	未使用
4	未使用
3	未使用
2	キーが押されていると1
1	未使用
0 (LSB)	キーが押されると，そのコードが1Aポートから読み込まれるまで1

表7-2　キーステータスのポート（1BH）

ビットNo.	内容	
7 (MSB)	ディスプレイの帰線期間中は0になっている	
6	未 使 用	
5	未 使 用	
4	TRIGGER	ともにONのとき0でOFFのとき1
3	RIGHT	
2	LEFT	
1	BACK	
0 (LSB)	FORWARD	

表7-3　ジョイスティックのポート(51Hポート)

キーコードは1AHから，キーステータスは1BH(表7-2)から読まれます．また，ジョイスティックは51H ポートに割り当てられていて，アクセスするときはCレジスタを使うIN 命令を使い，Bレジスタのビット0が1だとカーソルパドルとスペースバーまたはジョイスティック1が，Bレジスタのビット0が0だとジョイスティック2が選択されます(表7-3)．

7-2　応用周辺機器の自作

SMC-777にはバスラインを外部に引き出して使えるように，I／O Expansion が設けられています．したがって，これに自作のインターフェイスを接続し，SMC-777の処理対象を大幅に広げてやることもできるわけです．この節では，そうした自作という観点から考え，近い将来パソコンがHA (Home Automation)の主役になるであろうことも加味し，風呂の温度センサーの製作とSMC-777での制御を，参考のために紹介してみます．読者諸兄が以下の説明から，パソコンによる自作機器制御のアイデアなりとも手に入れてくだされば と思います．

(1) ハード

I/Oポートの割り付け

インターフェイスの拡張に当たっては，SMC-777の設計上，すでに利用されていたり利用予定のあるI／O ポートの使用は避けねばなりません．未使用のポートは66H～70H のみなので，ここを利用することにします．

サーミスタ

サーミスタとは，様々な元素を組み合わせて高温処理して作られた半導体で，精密温度測定や自動温度調整器，電圧制御など幅広く使われている半導体素子です．図7-4に示したように，サーミスタは温度が高くなるに従って電気抵抗が減少するので，サーミスタと直列に適当な抵抗を

接続してやると，温度変化によるサーミスタの抵抗変化を電圧変化として取り出すことができるようになります.

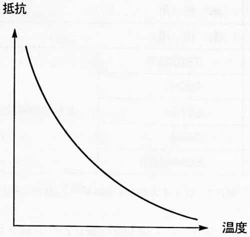


図7-4　サーミスタの温度と抵抗

温度センサーの構成と原理

図7-5に温度センサーの構成図を示します.構成は,温度センサー部のその出力を増幅するAmp,電圧を読むA／Dコンバータ，A／Dコンバータの出力を保持するラッチ回路，そしてA／D変換開始信号等をコントロールする，ステータスコントローラから成っています.

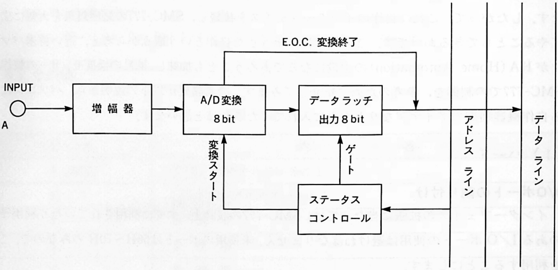


図7-5　温度センサーのブロック図

バスを引きだす

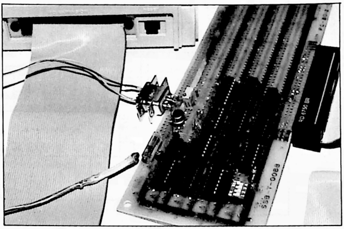
SMC-777からZ-80Aのバスラインを取り出すには，本体のケースを取りはずし，50ピンのI／O Expansion Connector（オス）を見つけ出します．表7-4に示したのが該当コネクタのピンアサインで，ここではフラットケーブルを使い，バスラインを本体外部に引き出しました.

1	GND	14	A13	27	GND	40	D3
2	+5V	15	A14	28	ϕ	41	D4
3	+5V	16	A15	29	GND	42	D5
4	+5V	17	A0	30	GND	43	D6
5	+5V	18	A1	31	$\overline{WR}$	44	D7
6	+5V	19	A2	32	$\overline{RD}$	45	RTRN
7	+5V	20	A3	33	$\overline{IORQ}$	46	RTRN
8	$\overline{WAIT}$	21	A4	34	$\overline{HALT}$	47	+12V
9	A8	22	A5	35	$\overline{INT}$	48	+12V
10	A9	23	A6	36	$\overline{RES}$	49	−12V
11	A10	24	A7	37	D0	50	$\overline{NMI}$
12	A11	25	GND	38	D1		
13	A12	26	$\overline{M1}$	39	D2		

表7-4 I/O Expansionのピンアサイン

温度センサ回路

図7－6に表示したものが温度センサー用インターフェイスの回路図です．チップセレクト回路は，70H（A／D コンバータスタート＆クリア），71H（ステータスリード），72H（データリード）をそれぞれセレクトする回路です．A／D 変換回路は，基準電圧を発生させる AD581JH，8 ビットのA／D コンバータ（AD7574JN），ラッチ（74LS373），そのほかこれらをコントロールする回路（74LS123，74LS74等）により構成されています．増幅回路の方は，NE5538Nとなっています．



自作の温度センサーをSMC-777に接続する

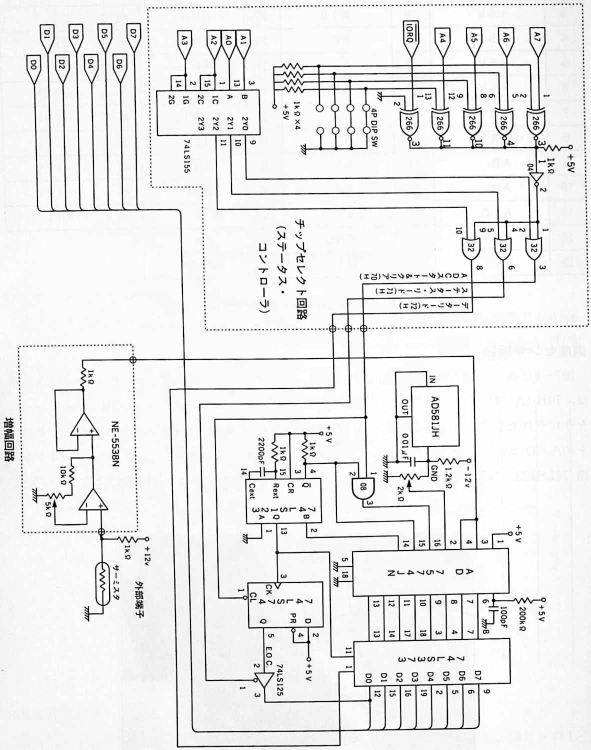


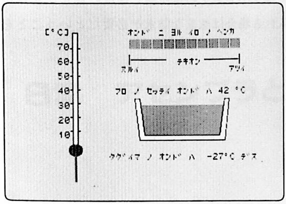
図7-6　温度センサーの回路図

(2) 温度センサー用プログラム

List 7-1

```
 10 '
 20 ' --- ｵﾌﾛ ｾﾝｻｰ ---
 30 *MAIN
 40  WIPE : CONSOLE 40 : LOCATE (8,10)
 50  INPUT "ｵﾌﾛ ﾉ ｾｯﾃｲ ｵﾝﾄﾞ ﾊ ? ",ASET%
 60  WIPE : CURSOR OFF
 70  FOR I=6 TO 15
 80    DEF PALETTE I,I,4,15-I
 90  NEXT
100  DEF PALETTE 0, 5,15, 6
110  DEF PALETTE 1, 0, 0, 0
120  DEF PALETTE 2,15, 0, 0
130  DEF PALETTE 3, 0,15,15
140  DEF PALETTE 4,15,15,15
150  DEF PALETTE 5, 0,14,14
160  CPALETTE OFF
170  GCLEAR 0
180  GCOLOR 0,0,0
190  GOSUB *TEMPBAR
200  GOSUB *GLIN
210  GOSUB *MSG
220  GOSUB *BATH
230  GOSUB *WTEMP
240 ' --- END MAIN ---
250 '
260 *TEMPBAR
270  FOR I=6 TO 15
280    TX=I*15+40 : TX1=I*15+53
290    BOXF(TX,35)-(TX1,45),I
300  NEXT
310  LINE(130,52)-(130,62),1
320  LINE(278,52)-(278,62),1
330  LINE(132,57)-(177,57),1
340  LINE(229,57)-(276,57),1
350  RETURN
360 '
370 *BATH
380  DRAW "S1C1m113,100W5,40W80,0"
390  DRAW "W5,-40W5,0W-5,45W-90,0"
400  DRAW "W-5,-45W5,0"
410  DRAW "w-2,2P4,1C3w3,5"
420  DRAW "W88,0w-4,4P3,1"
430  RETURN
440 '
450 *WTEMP
460  IMEM = 0
470  GOSUB *MEASURE
480  IF TMP<TMEM THEN *ERBAR
490  IIT=150-TMP*1.5
500  BOXF(61,150)-(63,IIT),2
510  I% = TMP/6+3
520  DEF PALETTE 3,I%,4,15-I%
530  TMEM = TMP
```

```
540  TE% = TMP
550  PRINT (29,20);TE%
560  IF ASET% <= TE% THEN GOSUB *MUSIC
570  GOTO *WTEMP
580  RETURN
590 '
600 *ERBAR
610  IT = 150-TMP*1.5
620  IT1 = 150-TMEM*1.5
630  BOXF(61,IT)-(63,IT1),5
640  GOTO 510
650 '
660 *GLIN
670  BOXF (59,28)-(65,150),5
680  BOXE (59,28)-(65,150),1
690  CIRCLE (62,156),8,,1
700  PAINT (63,155),2,1
710  FOR IY=135 TO 40 STEP -15
720    LINE(56,IY)-(59,IY),1
730    LINE(65,IY)-(68,IY),1
740  NEXT
750  CCOLOR 0
760  FOR I=17 TO 5 STEP -2
770    N=N+10
780    PRINT (5,I);N
790  NEXT
800  RETURN
810 '
820 *MSG
830  PRINT (3,3)  ;"[°C]"
840  PRINT (15,3) ;" ｵﾝﾄﾞ ﾆ ﾖﾙ "
850  PRINT (26,3) ;"ｲﾛ ﾉ ﾍﾝｶ "
860  PRINT (13,11);" ﾌﾛ ﾉ ｾｯﾃｲ ｵﾝﾄﾞ ﾊ    °C "
870  PRINT (31,11);ASET%
880  PRINT (15,8) ;"ﾇﾙｲ"
890  PRINT (33,8) ;"ｱﾂｲ"
900  PRINT (24,7) ;"ﾃｷｵﾝ"
910  PRINT (13,20);"ﾀﾀﾞｲﾏ ﾉ "
920  PRINT (21,20);"ｵﾝﾄﾞ ﾊ"
930  PRINT (32,20);"°C ﾃﾞｽ"
940  RETURN
950 '
960 *MEASURE
970  OUT &H70,0
980  A=INP(&H71)
990  IF (&H1 AND A)=0 THEN 980
1000 B=INP(&H72)
1010 TA=255-B
1020 TMP=TA/2.5-27
1030 RETURN
1040 '
1050 *MUSIC
1060 IF EOS=0 THEN RETURN
1070 SOUND "<8S1003L16G04C>P8"
1080 RETURN
1090 ' --- PROGRAM END ---
```



風呂の温度の表示

さて，インターフェイスができたら，次はプログラムを作らねばなりません．せっかくグラフィックス機能の豊富なSMC-777なのですから，これを利用しない手はありません．リスト7-1がそのプログラムです．実行画面の左の温度計の形をした部分で現在の湯温をアナログ的に表示し，右下のバスタブ（風呂桶のつもりだ，文句あるか！）のなかでは温度によってお湯の色が変化します．また，バスタブの下には現在の湯温をデジタル表示する部分も加えます．プログラム各ブロックの概略は，以下のとおりです．

- *MAIN　：グラフィックス画面のアウトラインの設定します
- *TEMPBAR：温度による，風呂のお湯の変化のモニターを描きます（実行画面右上）
- *BATH　：バスタブの外形を描きます
- *WTEMP　：測定された湯温を温度計のバーに示し，バスタブのお湯の色やデジタル表示を行います
- *ERBAR　：測定された温度が前回の測定値より小さいとき，温度計のバーを測定値まで消します
- *GLIN　：温度計の外形を描きます
- *MEASURE：A／D コンバータのコントロール及び，A／D コンバータの値を CPU に読み込ませます
- *MUSIC　：温度が設定値以上に上昇したとき，アラームを鳴らして警告します

以上がプログラムの主な働きです．このなかで最も重要な働きをするのが*MEASUREで，これを理解しておくとほかのインターフェイス自作のときにも役立つと思われるため，少々詳しく説明しておきます．まず，970行は70H ポートをセレクトし，A／D コンバータにクリアとスタートパルスを出力します．次の980行で，A／D 変換が終了したことを確認するため71H ポートのデータを変数 A に読み込み，990行で変数 A の LSB が 1 であれば，A／D 変換は終了したことになります（変換終了が得られない場合は980行に戻ります）．1000行はA／D変換されたデータを72H ポートから変数Bに読み込むもので，1010行，1020行で変数Bの内容を温度に変換しています．

以上が温度センサーの概略ですが，ここではサーミスタの抵抗値と温度の補正が正確にはされていないということと，サーミスタを水に入れる場合は厳重な防水が必要だということを記憶しておいてください．

8章　アセンブラとSony Filer

パソコンに何かをやらせるとき，BASICだけでプログラムを作っていては，スピードに関して満足なものが得られないことが多々あります．また，BASICでは記述不可能なこともあります．そういったときに，新たなコマンドなり関数を機械語で作ってBASICを拡張することがよくあります．この章ではこのようなBASICを拡張するための機械語ルーチンに限らず，SMC-777で機械語を扱うこと全般について，解説してゆきます．

8-1　アセンブラ

(1) 777 Assemblerとは

777 BASICではLINKコマンドを使うことによりコマンドの拡張ができるのは何度も触れたとおりです．リンクパッケージとして777のシステムディスクにはCOPY．PAC，PIC．PAC，EFFECT．PACの3つが入っていますが，777 Assemblerはこういったリンクパッケージを作るためのアセンブラなのです．

ニーモニック

ソースプログラムにはザイログニーモニックともインテルニーモックとも似ても似つかないANN（Assembler in Natural Notation）表記というソニーのオリジナルのニーモニックを使います．ザイログニーモニックとANN表記の対応表をAPPENDIXに示しますが，主な規則は以下のとおりです．

・LDは＝を使ってBASICの代入文のように表記する

LD A，B　　　→ a＝b

LD (IX＋d)，C　→ (ix＋d)＝c

・演算の対応

ADD　→ ＋

SUB　→ −

AND　→ &

OR　→ |

XOR　→ ^

NOT　→ ~

・結果がフラグ以外にも残る演算も，そのレジスタに対する代入のように表記する

XOR A　　　→a＝a ^ a

ADD HL，SP →hl＝hl＋sp

CP (HL)　　→ a −(hl)

・フラグの対応

Carry　→ @

Zero　→ 0

Plus　→ ＋

Minus　→ −

PE　→ e

PO　→ o

従って，

ADC HL，BC →hl＝hl＋bc@

SCF　　　　→ @＝1

CCF　　　　→ ~@

・絶対ジャンプは＝＞相対ジャンプは−＞で表記する

JP LABEL　　→ ＝＞label

JP NC，LABEL→ ＝＞label，~@

JR Z，LABEL → −＞label，0

DJNZ LABEL → ~＞label

・CALLは[，RETは]で表記する

CALL ABC　→] abc

RET PE　　→ [,e

RST 8H　　→] '08

擬似命令

777 Assemblerでは，実際の処理の前にorg，entry，startが，この順でどれも欠けずになくてはなりません．そのほか，注意を要する擬似命令について以下に説明します．

entry

オペランドに,CALLで呼び出すときに使うコマンド名を書く.そのルーチンのエントリポイントには,コマンド名をラベルとして付ける.entryはそのリンクパッケージ中のエントリ数だけ必要.

start

オペランドに，そのリンクパッケージがリンクされた直後に実行すべきコマンド列（スタートアップコマンド）を〝でくくって書く．スタートアップコマンドを設定しないときはオペランドを付けず，ただstartと書く．

db

使い方はおおむね一般のアセンブラと同じだが,〝でくくった文字列をオペランドとすることもできる．たとえば，

db〝Noriko Haratake", 0

これらのほかに擬似命令としてequ（ラベルの定義），dw（16ビット値の定義），ds（nバイトの領域確保），endがありますが，これらは一般のアセンブラと同じです．

引数

5章でも書いたように，LINKで拡張するコマンドの書式は，

CALL　<コマンド名>（<引数1>，<引数2>･･･）

または，

_<コマンド名>（<引数1>，<引数2>･･･）

で，もともと備わっているコマンドと同じように引数を持たせることができます．引数を持たなくてももちろんかまいません．

引数の設定は擬似命令entryで行ないます．entryの書式は，

entry　<コマンド名>（<引数1の型>，<引数2の型>･･･）

です．また，<引数の型>の一覧を表8－1に示します．またこれらを[　]で囲むことで，その引数が省略可能であることを示すことができます．例えば，CBSというコマンドに2つ引数を持たせるとして，1番目は整数型，2番目は文字型で省略可能とすると，entryは，

entry cbs (int, [str])

となります.

書　式	内　　　　　容
BYT	1バイト整数型
INT	2バイト整数型
ADR	INTとだいたい同じだが32768～65535が －32768～－1に変換される
FLT	浮動小数点型
STR	文字型
ANY	型チェックなし

表8-1　引数の型の書式

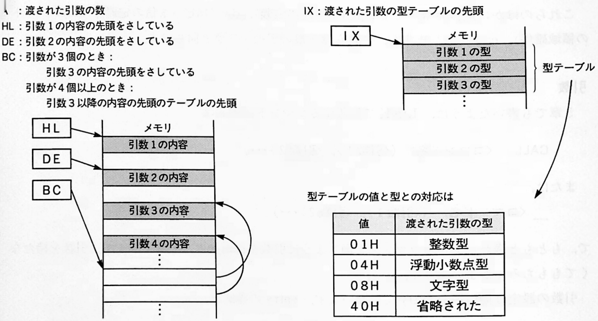


図8-1　拡張コマンドにエントリしたときのレジスタの内容

拡張コマンドがCALLで呼び出されたとき，レジスタに入っている値は図8－1のようになっています．実際の引数はentryに書かれている型と同じ型でなくてもよく（本来BASICのデータの型としてはadr，bytなどはありません），自動的に型変換をしてくれます．このとき，どうして

も変換できない場合（文字型が要求されているのに数値が渡されたとか，bytに256以上が渡されたとか）は，Type mismatchやOverflowなどのエラーとなります．

またentryに書いてある型と同じ型の変数が引数として渡された場合は，HL, DE, BCなどが指し示すのは，その変数の内容そのものですから，これを利用して値を返すことができます．ただし，この場合同じ型のという制限がありますから，数値を返す場合は，entryのほうにはanyで受けておいて(IX＋d)で実数型，整数型どちらの変数が渡されたかを調べ，最後に値を返すとき適当に型を変換しなければならず，結構大変です．

ソースの作成

ソースプログラムはBASICのテキストとして作成します．基本的に普通のBASICのテキストと同じように入力すればいいのですが，すべての行の頭に '(アポストロフィ）をつけて注釈として入力した方が無難です(例えば 0e00hは 0.hとなってしまう)．そのほか注意すべき点として次のようなものが挙げられます．

・大文字と小文字

〃でくくってあるもの以外は大文字，小文字は全く区別されない．

・ラベル

100文字以内の英数字の列で，先頭は英字でなければならない．equの前に書くときと，オペランドとするとき以外（つまりその場所にラベルを付けるとき）は，ラベルのうしろに：を付けなければならない．

・文字列

文字列は〃で囲むが，〃を文字列に含めたいときは〃を2つ重ねる．

・オペランド

普通のオペコード（オペコードとオペラントが明確ではないが）に対するオペランドとしては，数字かラベルしか許されない．たとえば，

```
CP "B"
```

の意味で

```
a — "B"
```

とするのは許されないし，

```
hl＝label＋1
```

というように計算式をオペランドとすることもできない．文字列がオペランドとして許されるのはstart，dbのみである．

・数字

2進数はうしろにbを，16進数はhを付ける．16進数が英字で始まるときは，ラベルと区別するため，先頭に0をつける．

・注釈

行の；以降は注釈として解釈される.

ソースの入力が終ったら拡張子にANNを付けてディスクにセーブしますが，そのときSAVE／Aを使ってアスキー形式で行います.

アセンブル

システムメニューに戻ってASSEMコマンドによりアセンブルします.

ASSEM 〈ファイル名〉

と入力するわけですが，このときファイル名に拡張子を付けないと，拡張子として，ANNが自動的に補われます．これで，

〈ASSEMに与えたファイル名のプライマリネーム〉.PAC

というファイル名のリンクパッケージが，ソースの入っているディスクに作られます.

アセンブラを起動するときにファイル名のあとに／L,／T,／Aのうち，いくつかを付けると，次のようになります.

・／L

PACと同時にリスティングファイルである．LST（オブジェクトのリストの右にソースをくっつけたもの）が作成されます.

・／T

PACと同時にテストファイルである．TSTが作成されます．．PACには実際にメモリへロードされるオブジェクトの前に余分なものが13hバイトついているため，Debuggerでロードしてデバッグするときは，このテストファイルを用います.

・／A

通常リンクパッケージは，リンクされるときにリロケートされてBASICインタープリタの最後にロードされます（これをリロケータブル・リンクパッケージファイルと呼びます).／Aを指定するとロードする位置が始めから決まっているアブソリュート・リンクパッケージファイルが作成されます．アブソリュート・ファイルを作成するとき以外はorgでどう指定しても，100hがオリジンになってしまいます.

／L,／T,／Aのうち複数を指定するとき，たとえばリスティングファイルとテストファイルを同時に作成したい場合は，

／L／T　／T／L　／TL　／LT

のうちどれを用いてもかまいません.

そのほかの注意

当然のことですが，拡張コマンドの実行を終るときは [を使い，従って，スタックのレベルがこのとき合っていなければなりません．リンクパッケージ内でスタックは8バイト(4レベル)が保証されているのみで，これ以上スタックを使いたいときは，自分でどうにかしなければなりません.

通常リンクされるときにはリンクパッケージがリロケートされるわけですが，このとき，どういうところが書き換えられてどういうところが書き換えられないのか，というのが気になるところです．書き換えられるのはequで宣言されたラベル以外を使っているところで，こういったラベル以外を使っているものは，

＝＞label]label

のほか，

hl＝label

等も書き換えられます.

ところで，機械語ルーチンを作るときBASICインタープリタ内部のルーチンを利用することも多々あります．しかし，ソニーの方針としてインタープリタの中身はルーチンのエントリもワークエリアも変えることがあるそうなので，あとあとのことを考えて，そういったものは使わない方が無難でしょう．ただし，シャドウROMに関してはルーチンのエントリは変えないそうです(変えるとソニーの人も困ってしまうそうな).

(2) 実行例

加算

List 8-1

```
10 '
20 ' ASSEM & BASIC
30 ' LINK 'KASAN.PAC'
40 '
50 DEF INT A-C
60 A=127
70 _KASAN(A,511,B)
80 PRINT '127+511=';B
90 END
```

List 8-2

```
 10 ';==================================================
 20 ';      Sample to explain the 'ASSEM', KASAN
 30 ';==================================================
 40 ';
 50 '           org     100h
 60 '           entry   kasan(int,int,int)
 70 '           start
 80 ';
 90 'kasan:     a=(de)
100 '           a=a+(hl)
110 '           (bc)=a
120 '           de+
130 '           hl+
140 '           bc+
150 '           a=(de)
160 '           a=a+(hl)@
170 '           (bc)=a
180 '           [
190 ';
200 '           end
```

では，具体的に短いプログラムを提示して，Assembler を用いてどのようなことをするのかを，順を追って説明してゆくことにしましょう．リスト8-1にBASICのメインルーチン，リスト8-2にサブルーチンをANN表記で書いたソースプログラムを提示します．実際のプログラムの手順では，全体の構成を決めたあとにまずリスト8-2を入力し，エディットモードでデバックを済ませたあとに，

SAVE／A "KASAN.ANN"

としてソースプログラムをディスクに格納し，その後にシステムメニューに戻し，カーソルパドルで，

[ASSEM]　[KASAN.ANN]

をそれぞれ選択して[↵]を押します(もちろんキーボードから直接入力してもよいが，手間がかかります)．アセンブル中は，

PASS I

PASS II

と表示され，アセンブルの終了を知らせる，

error：0

Hit any key to continue………

で終ります．このときソースプログラムにエラーがあると，順次エラーメッセージが表示され，最後にエラーが合計いくつあったかまで表示されるので，もう一度BASIC上にロードして，デバッグしてから上記の手順をくり返すことになります．

アセンブル終了後，再びBASICを起動させてメインルーチンを入力し，バグがないことを確認したなら，

LINK "KASAN"

で機械語サブルーチンとメインルーチンがリンクし，プログラムが走ります．まァ，プログラムを実行してみたら，しょーもないことをしてやがる，とお思いになる方も多いでしょう．しかし，これも勉強のうちでして，Assemblerに慣れるための大切な要素が含まれているのです．リスト8-1の行番号70を見てください．

70 _KASAN (A, 511, B)

アンダーバーは何度も出てきたようにCALLと同じで，拡張コマンドを実行するための命令ですが，ここではAの値と511をサブルーチンに引き渡し，かつBに対してサブルーチンからメインルーチンへと値を引き渡しています．実はこうした値の引き渡し方法の充実が777 Assemblerの特長となっており，機械語サブルーチンの使い勝手を高めている理由でもあります．

ハエが飛ぶ

実行例に足し算なんか使いやがって，と思っておられる方々のために，機械語サブルーチンを使うと結構いろんな事ができるのだという一端を，例を用いて紹介してみましょう．特にゲームプログラム等に有効な手法ですので，ちょっと賢いAssemblerをよく呑みこみ，積極的に活用してください．まず，メインルーチンをリスト8-3に，また拡張コマンドFLYMOVEをリスト8-4に示します．アセンブルからLINKにかけての手順は前述のとおりですので，さっそく入力してみてください．

List 8-3

```
 10 '
 20 '   ASSEM & BASIC (2)
 30 '
 40 '   LINK 'PPT.PAC'
 50 DEF INT A-Z
 60 GCLEAR 3 : CCLEAR : CURSOR OFF
 70 FOR Y=0 TO 23
 80   FOR X=150 TO 0 STEP -1
 90    _FLYMOVE(X,Y,1)
100     FOR I=0 TO 10 : NEXT
110    _FLYMOVE(X,Y,0)
120   NEXT X
130 NEXT Y
140 WIPE
```

List 8-4

```
 10 ';===============================================
 20 ';    Sample to explain the 'ASSEM'#2, flymove
 30 ';===============================================
 40 ';
 50 '          org     100h
 60 '          entry   flymove(int,int,int)
 70 '          start
 80 ';
 90 'flymove:  a=(bc)
100 '          (pnum)=a
110 '          c=(hl)
120 '          hl+
130 '          b=(hl)
140 '          a=(de)
150 '          l=a
160 '          de+
170 '          a=(de)
180 '          h=a
190 '          hl=hl+hl
200 '          hl=hl+hl
210 '          hl=hl+hl
220 '          hl=hl+hl
230 '          hl=hl+hl
240 '          e=l
250 '          d=h
260 '          hl=hl+hl
270 '          hl=hl+hl
280 '          hl=hl+de
290 '          hl=hl+bc
300 '          bc=8000h
310 '          hl=hl+bc
320 '          (vdr)=hl
330 '          a=(pnum)
340 '          de=gpat1
350 '          a-1
360 '          ->ppt1,0
370 '          de=gpat2
380 '          a-2
390 '          ->ppt1,0
400 '          de=gpat0
410 'ppt1:     h=8
420 'ppt3:     #=hl
430 '          hl=(vdr)
440 '          b=1
450 '          c=h
460 '          l=4
470 'ppt2:     a=(de)
480 '          p(c)=a
490 '          #=de
500 '          d=c
510 '          e=b
520 '          de+
530 '          b=e
540 '          c=d
550 '          de=#
560 '          de+
570 '          l-
580 '          ->ppt2,~0
```

```
590 '           hl=(vdr)
600 '           bc=1000h
610 '           hl=hl+bc
620 '           (vdr)=hl
630 '           hl=#
640 '           h-
650 '           ->ppt3,~0
660 '           [
670 'pnum:      db     0
680 'vdr:       dw     0
690 ';
700 'gpat0:     db    33h,33h,33h,33h
710 '           db    33h,33h,33h,33h
720 '           db    33h,33h,33h,33h
730 '           db    33h,33h,33h,33h
740 '           db    33h,33h,33h,33h
750 '           db    33h,33h,33h,33h
760 '           db    33h,33h,33h,33h
770 '           db    33h,33h,33h,33h
780 'gpat1:     db    33h, 33h, 33h,0f3h
790 '           db    33h, 33h,0ffh,0ffh
800 '           db    30h, 3fh,0ffh,0ffh
810 '           db     4h,  8h, 88h, 33h
820 '           db    40h,  3h, 88h, 88h
830 '           db    33h,  3h, 38h, 83h
840 '           db     0h, 30h, 33h, 33h
850 '           db    33h, 30h, 33h, 33h
860 'gpat2:     db    0,0,0,0
870 '           db    0,0,0,0
880 '           db    0,0,0,0
890 '           db    0,0,0,0
900 '           db    0,0,0,0
910 '           db    0,0,0,0
920 '           db    0,0,0,0
930 '           db    0,0,0,0
940 ';
950 '           end
```

では，説明を加えましょう．このプログラムは，サブルーチン名が示しているように，ハエの飛行を表現するものです．メインルーチンは，サブルーチンの持っているハエのパターンを右から左に移動し，ボーダーエリアにつきあたったらY軸を8ドット分下げて再び同じパターンで移動させる役割りを負っています．またサブルーチンの方はGRAM上にx, yで指定された位置に，指定された8×8ドットのパターンを書き込むものです．

- 90〜180 ：引数x，y，hの読込み
- 190〜320 ：GRAMのアドレスの算出
- 330〜400 ：指定パターン番号から，パターンの入っているアドレスの算出
- 410〜650 ：パターンをGRAMに書き込む

というのが拡張コマンドの主な概略ですが，ハードウェアの知識が必要なため7章にあらかじめ

目を通しておくとわかり易いと思います．ハエのパターンは図8－2に示したようなもので，GRAM上に引数x，yで指定された位置にNで指定されたパターンで書き込まれます．このとき，xはX座標を指定しており範囲は0～156までの整数で，1ステップがグラフィックスの2ドット相当．yはY座標を指定し，0～24までの整数で，1ステップが8ドット相当となります．NはGRAM上に書き込むパターンの番号で，0～2までの整数で，このパターンで言うと，

N＝0：水色のみ
N＝1：ハエのみ
N＝2：黒のみ

となっており，ハエが羽を動かしながら画面上を飛んでいるように見せる役割りを担っています．また，このプログラムで用いた座標では，GRAMのアドレスとの関係は，

GRAM アドレス ＝ Y×160＋X＋8000H
（Y：y軸，X：x軸，8000H：GRAMの初期アドレス）

を用いて座標変換をしています．

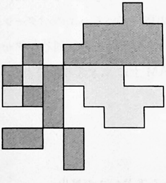
						灰	
				灰	灰	灰	灰
	黒		灰	灰	灰	灰	灰
黒	赤	黒	茶	茶	茶		
赤	黒	黒		茶	茶	茶	茶
		黒			茶	茶	
黒	黒		黒				
			黒				

（b）

3	3	3	3	3	3	F	3
3	3	3	3	F	F	F	F
3	0	3	F	F	F	F	F
0	4	0	8	8	8	3	3
4	0	0	3	8	8	8	8
3	3	0	3	3	8	8	3
0	0	3	0	3	3	3	3
3	3	3	0	3	3	3	3

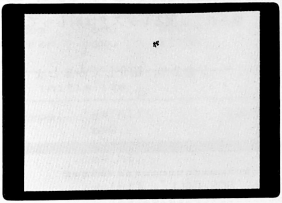
（c）

3……水色（バック）
0……黒
4……赤
8……茶
F……灰



（a）

図8-2　ハエ(GPAT 1)のパターン



ハエが飛ぶ

8-2 DOSコール

SMC-777では，システム内部（特にI/O関係）のルーチンが5番地及び8番地へのDOSコールとして公開されていて，みずからの手でリンクパッケージを作るような場合に非常に役に立ちます．このSony Filerの特徴である5番地コール，8番地コールを十分に有効活用することができれば，いままでのパソコンとはひと味違ったこともできるようになります．ただし，こうした知識は，少なくともパソコンに関する素養を多少持つ人のものであるとは思えますが……．

(1) なぜDOSコールなのか

今，BASICのリンクパッケージを自分で作らなければならないと仮定し，かつそのリンクパッケージのなかではデータをキーボードから入力したり，画面に何かを入力したりディスクを読んだりしなければならないとします．こういった入出力関係のルーチン（I/Oルーチン）を自分で書こうとするならハードウェアの知識が必要ですし，また一般的に考えれば非常に厄介なことでもあります．

ところでBASICにはINPUTやPRINTやSAVEなどがあるわけですから，BASIC自体にとってもI/Oルーチンは必要なものであり，またLogoやMEMOについても同様のことが言えるのです．そこでI/Oルーチンやそのほかのよく使いそうなルーチンをメモリに常駐させておき，それを皆で使ってはどうか，という考え方が出てきます．そうすると，キー入力の方法やファイルの構造などが統一されるという利益も生まれてきます．このようなことを前提に，SMC-777には5番地コールと8番地コール（この2つをまとめてDOSコールと呼ぶ）というものが用意されており，これらがI/O関係や，そのほかよく必要となるようなことを行ってくれます(取扱説明書の60頁以降参照)．たとえばCレジスタに2を入れて5番地をコールすると，Eレジスタの値のキャラクタコードを持つ文字を画面に出力してくれますし，Cレジスタに5を入れてコールすると，同様にプリンタに出力してくれます．

5番地コール，8番地コールは，必要に応じて条件を整えたあと，使いたい機能の番号（ファ

ンクションナンバー）をCレジスタに入れて呼ぶように定められています．そして，これらのルーチンではレジスタは全く保存されませんから，壊されては困るレジスタは前もって退避させておく必要があります．

それでは実際にDOSコールを使ったリンクパッケージを2つ，紹介してみましょう．

（2）サンプル・プログラム

List 8-5

```
 10 ´;=============================================
 20 ´;        Sample to explain ´DOS CALL´
 30 ´;=============================================
 40 ´;
 50 ´          org     100h
 60 ´          entry   greatsmc
 70 ´          start
 80 ´;
 90 ´greatsmc:e=14     ;^CLR
100 ´          c=2
110 ´          J´08
120 ´;
130 ´          d=32    ;width check
140 ´          c=21
150 ´          J´08
160 ´          a=a¦a
170 ´          ->grlabel1,0
180 ´          l=8
190 ´          ->grlabel2
200 ´grlabel1:l=28
210 ´grlabel2:h=12
220 ´          d=0     ;locate
230 ´          c=21
240 ´          J´08
250 ´;
260 ´          de=grstring
270 ´          c=9     ;print out a string
280 ´          J´08
290 ´          [
300 ´;
310 ´grstring:db      ´SMC-777 is excellent!!´,0
320 ´;
330 ´          end
```

List 8-6

```
 10 ´;=============================================
 20 ´;       ´type´ to printer,ltype
 30 ´;=============================================
 40 ´;
 50 ´          org     100h
 60 ´          entry   ltype(str)
 70 ´          start
```

```
 80 ';
 90 'bdos       equ       0005h
100 'error      equ       0115h
110 ';
120 ';--------------------------------------------
130 ';  initialize
140 ';--------------------------------------------
150 'ltype:     c=(hl)                 ;setfcb
160 '           b=0
170 '           hl+
180 '           e=(hl)
190 '           hl+
200 '           d=(hl)
210 '           hl=005ch
220 '           (hl)=0
230 '           hl+
240 '           de:hl
250 '           {(de+)=(hl+)}
260 '           de:hl
270 '           (hl)=0
280 ';
290 '           de=005ch
300 '           c=15                   ;open file
310 '           Jbdos
320 '           a+
330 '           ->ltopnorm,~0
340 '           a=10                   ;file not found
350 '           Jerror
360 ';
370 'ltopnorm:de=0080h                 ;open normally
380 '           c=26                   ;set DMA area
390 '           Jbdos
400 ';
410 ';--------------------------------------------
420 ';      main
430 ';--------------------------------------------
440 'ltloop2: de=005ch
450 '           c=20                   ;read disk
460 '           Jbdos
470 '           a+
480 '           ->ltcrlf,0             ;return if eof
490 ';
500 '           hl=0080h
510 '           b=80h
520 ';
530 'ltloop1:  #=bc
540 '           #=hl
550 '           c=6                    ;direct key input
560 '           e=0ffh
570 '           J'08
580 '           a-1bh                  ;chr$(1bh)=ESC
590 '           ->ltend,0
600 '           hl=#
610 '           #=hl
620 'ltnkeyin:a=(hl)
630 '           a-1ah
640 '           ->ltend,0
650 '           e=a
```

```
660 '          c=5               ;print@1
670 '          Jbdos
680 '          hl=#
690 '          bc=#
700 '          hl+
710 '          ~>ltloop1
720 '          ->ltloop2
730 ';
740 'ltend:    bc=#
750 '          hl=#
760 'ltcrlf:   e=13     ;CR
770 '          c=5
780 '          J'08
790 '          e=10     ;LF
800 '          c=5
810 '          J'08
820 '          [
830 ';
840 '          end
```

リスト8-5は画面を消して，その中央に「SMC-777 is excellent ！ ！」と表示するリンクパッケージのソースリストです．「＿GREATSMC」を実行しますと，画面が横80桁でも40桁でも中央に「SMC-777 is excellent ！ ！」と表示されます．要点を説明してゆきましょう．

90～110：画面を消します．CTRL+CLRのキャラクタコードは14です．

130～150：画面の桁数を調べています．80行のときは Acc に00が，40行のときは00以外が入って戻ってきます．8番地コールの21番はサブナンバーとしてDレジスタにも値を設定し，Cレジスタと Dレジスタの値で機能が決ります．

160～240：80行のときは（28，12）に，40行のときは（8，12）にカーソルを移動させます．

260～280：カーソルの位置から「SMC-777 is excellent ！ ！」と表示します．8番地コールの9番は DE レジスタの指すアドレスから00が見つかるまでを表示します．そのため310行のメッセージの最後に，0が付いているのです．

リスト8-6は TYPE と同じことをプリンタへ行う，いわばLTYPE と呼べるものです．＿LTYPE (<ファイル名>)で，そのファイルをアスキーファイルとしてプリンタへ出力します．ただしここで指定する<ファイル名>は，BASIC で通常使うファイル名と少し異っています．まずプライマリーネームは必ず8文字でなければならず（8文字未満のときはうしろにスペースを加えて8文字にします)そのつぎにピリオドを入れずに拡張子を必ず3文字書きます．そして英文字は大文字でなければなりません．例えば〝LTYPE.ANN〟は〝LTYPE␣␣␣ANN〟（␣はスペースのしるし）とします．BASIC で普通に使うファイル名で指定しようとすると，プログラムがこの3倍以上になってしまい，主な目的である DOS コールの説明には若干不適当となると

考えてこうしています．プログラムの要点を説明しましょう．

160～260：引数として与えられたファイル名をFCBに書き込みます．ここで言うFCBとはSony Filerがファイル名を扱うときに使う33バイトのワークエリアのようなもので，このプログラムは005CH番地から設定しています．ファイルを読み込むときにFCBに対してしなければならないことは，1バイト目を00にし，2バイト目から9バイト目にファイルのプライマリーネームを書き込むことと，10バイト目から12バイト目に拡張子を書き込み，13バイト目に00Hを書き込むことです．

280～300：ファイルをオープンします．DEにはFCBの先頭アドレスを入れます．160～260行で書き込んだ名前のファイルが見つからなかったときはFFHが，正常にオープンできたときは00HがAccに入ります．

360～380：Sony Filerではディスクの読み書きを128バイトごとに行いますが，読み書きに使うバッファ（DMAエリアと呼びます）を0080H番地から128バイトに設定しています．ディスクから読み込むときはこのバッファにディスクから読んだものが書き込まれ，書き込むときはこのバッファの内容がディスクに書き込まれます．

440～460：ディスクから1レコード（128バイト）読み込みます．その際自動的にさきに読み込んだレコードの次のレコードを読み込んでくれます．もう読み込むレコードがなかったときはFFHが，正常に読み込めたときは00HがAccに入ります．

500～710：0080番地から128バイトをプリンタに出力します．
550～590行で ESC キーが押されていたら処理を中断するようにしてあります．550～570行でキーが押されたかどうかを調べ，押されていたら580～590行でそれがESCかどうか見ています．ただし ESC を押しても止まるまで時間がかかりますし，止まらないこともあります．また630～640行でファイルの終りのしるしである1AHを検出したら処理を終るようにしてあります．アスキーファイルは必ずこの1AHで終っていますから，本来470～480行の読み込むレコードがなかったときの処置はしなくてもよいのですが，間違って（このプログラムをいじめるために？）アスキーファイルでないものに対してLTYPEを実行してしまったときのことを考えて，こうしてあります．

740～820：処理を終るに当たってCR／LFを出力しています．740～750行は530～540行のPUSHの埋め合わせをするためのものです．

ちなみにファイルを読み込んだだけのときは，それをクローズする必要はないので，このプログラムでもしていません．

(3) 5番地と8番地の違い

ところで，DOS コールに5番地コールと8番地コールの2種類があるのはなぜでしょうか．また5番地コールと8番地コールの0番から11番までは重複しているように見えますが，これはどういうわけなのでしょうか．詳細は避けますが，この2つはできた背景が違うのです．そして双方の0～11番の本体（本当に仕事をしているところ）はシャドウROM内にありますが，正真正銘同じものです．7番と10番は双方で機能が違いますが，これはその本体がどこから呼ばれたかで機能を変えているのです．したがって，8番地をコールするときRST 8 (ANNでは]'08)を使えば1バイトで済みますし実行も速いですから，5番地でも8番地でもよい場合は8番地をコールしたほうがよいでしょう．

DOS コールはリンクパッケージを作るときの便宜を図るためだけのものではありませんが，やはり一般のユーザーにとっては，これが大きな意味を持つと思います．従来の多くのBASICマシーンでは，I／O関係に限らずシステム内部のルーチンは全くメーカーからは公開されませんでした．ですからI／O関係を含む機械語ルーチンを作るには，システムを自分で解読するか，すべて自分で作るか，さもなくば誰かが解読してくれるのを待つしかありませんでした．前者の2つには非常なまでの苦労を要します(これがうれしいという人も，いるにはいますが)．そうした面で見た場合には，SMC-777におけるこのような設計思想は非常にユーザーライクなものであり，マニアの高度な期待に答えているものと言えるでしょう．

8-3　Sony Filerのコマンドを作る

(1) Sony Filerとは

SMC-777はDOSとしてSony Filerというオリジナルのものを採用しています．これはBDOSコール(先の5番地コール)のレベルでCP／M Version1.4と多くの互換性を持ち，ファイル構造に関してはSMC-70のCP／M Version2.2と完全な互換性があります．つまりSMC-70のファイルは全く問題なく読むことができるのです．

Sony Filerのメモリマップはおおまかにいって図8-2のようになっています．0000H～00FFH，FC00H～FFFFHまではシステムのワークエリア，E900H～FBFFHはファイルの管理や画面，キーボード，プリンタとの入出力を行うサブルーチンが入っている部分で，DOSと呼ばれます．このDOSの部分はメモリに常駐していて，Sony Filerがコールドスタートしたとき（リセットされたとき）読み込まれるだけですので，不用意に変更してはなりません．CP／MのBDOSとBIOSを併せたような機能をもつところなのです．UIFというのはUser Interfaceの略で，システムメニューの処理を行ないます．CP／MのCCP (Console Command Processor)にあたるものです．UIFの部分はシステムメニューに戻るとき（ウォームスタート）ディスクから読み込まれますから壊してもかまいません．

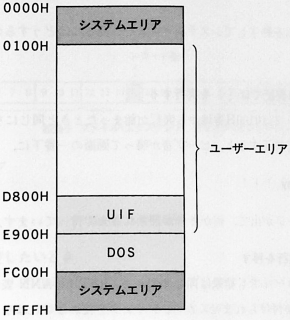


図8-3 Sony Filerのメモリマップ

シスメテムメニューの左，つまり「Command」という表示の下に出てくるものをファイラーのコマンドと呼びます．このなかで，TYPE, PRINT, ERASE, RENAME の4つはUIFのところにその処理ルーチンがあるビルトインコマンドで，その他は実行する際ディスクから読み込まれるトランジェントコマンドです．トランジェントコマンドが選択されるとUIFはそれを0100H番地以降に読み込み，読み込みが終ると0100H番地へジャンプします．

CP／M でも同様ですがSony Filer は1論理セクタが128バイトです（ディスクが物理的にどうであろうと，DOS コールのディスクアクセスでは128バイトが最小単位であるということ）．そしてファイルが正確に何バイトの大きさかが格納されるような特別な領域はなく，基本的にファイルは128バイト単位の大きさです．そこで正確に何バイトかを示したい場合はファイルのなかにその情報が入っていなければなりません．アスキーファイルの場合は最後に1AH のキャラクタを書き込むことでそれを行なっています．トランジェントコマンドのファイルには，そのようなものがなく，ファイルの大きさは128バイト単位ということになります．また，Sony Filer では拡張子に .EXE が付いているものがトランジェントコマンドと見なされます(CP／M では .COM)．

（2）コーディングにあたって

ここまでの説明で，トランジェントコマンドは0100H番地から始まる機械語プログラムで，拡張子に .EXE が付いているものだということは，わかっていただけたと思います．しかし，これだけの知識ではプログラムのコーディングもできません．

コマンドの終了

まず，プログラムを終了してシステムメニューに戻るにはどうするのか．これには，いくつも方法があります．

・**RET命令（ANN表記では［）を実行する**

もちろんスタックは0100H番地から実行が始まったときと同じになっていなければなりません．こうやって終了すると，ビープ音が鳴って画面の一番下に，

Hit any key ！！！

というメッセージが出て，何かキーが押されるまで待っています．

・**0000H番地へ実行を移す**

ジャンプでもコールでも結果は同じですから，RST 0（ANN表記では］'00)が短くてよい．RSTには条件が付けられませんから，ジャンプを使わなければならないときもありますが．

・**Cレジスタに0を入れて5番地または8番地へ実行を移す**

結果は0000H番地へ実行を移したときと同じです．

・**シャドウROMにバンクを切り換えて，0000番地へ実行を移す**

これまでの3つはいずれもウォームスタートでしたが，これはSony Filerのコールドスタートです．E900H番地以降の部分を壊してしまったときなどは，こうしなければなりません．

ファイルの管理

前項で少し触れましたが，Sony Filerではディスクのアクセスに際して，FCB (File Control Block)と呼ばれる33バイトのワークエリアを使います．FCBの構造を図8－4に示しますが，普通にシーケンシャルなアクセスをするだけなら，ファイルをオープンするときにバイト0からバイト12までを書き込めばあとはユーザーがこれを操作する必要はありません．このFCBは，オープンされているファイルごとに必要で，置いてある場所はどこでもかまいません．そこで，ファイルをアクセスするとき(5番地コールの15番から23番まで)は，DEレジスタにアクセスしたいファイルのFCBの先頭番地を入れておきます．ところで，UIFもTYPE，PRINT等，ファイルをアクセスしますし，またトランジェントコマンドをロードしたりもしますから，FCBを作ります．それは005CH～007CHにあり，トランジェントコマンド中でも，ファイルをひとつしかオープンしないのなら，通常ここにFCBを置きます．

ディクスに読み書きをするとき128バイトのバッファが必要となりますが（1論理セクタが128バイトだから)，これをDMAエリアと呼びます．FCBは5番地をコールする際いちいちDEに入れておくのに対し，DMAエリアは5番地コールの26番で変更するまでは変りません．UIFは0080H～00FFHをDMAに使っていますから，トランジェントコマンド中でもここを使うことが

多くなります．

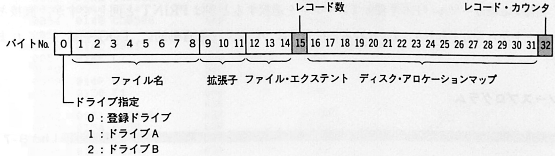


図8-4　FCBの構造

コマンドにファイルを渡したいとき

「ASSEM SOURCE」，「PRINT SOURCE. LST」等コマンドにファイル名を渡すときがよくあります．システムメニューからコマンドを入力したとき（カーソルパドルで選択しても，直接入力しても同じです），1つ目のファイル名は005CH～0067H，2つ目のファイル名(ファイルを2つ指定することは，カーソルパドルではできません)は006CH～0077Hにそれぞれ，FCBのバイト0からバイト11に書き込む形式で書き込まれます．たとえばトランジェントコマンドとして「TRANSIEN」というものがあったとして，「TRANSIEN ABCD B：EFGHI.JKL」と入力したとすると，

```
005C          00
005D～0067    "ABCD␣␣␣␣␣␣␣"
0068          02
006C～0077    "EFGHI␣␣␣JKL"
```

というように書き込まれます．

(3) LOGOPRNT

それでは実際の操作を実例を挙げて説明してゆきましょう．実例としてはLogoのファイルをプリントアウトするためのLOGOPRNTというものを作ります．

プログラムの仕様

Logoのプログラムのプリントアウトはセーブしたものを（Logoのセーブはすべてアスキー形式でおこなわれますから）システムメニューのPRINTを使って行うことが多いのですが，その際プロシージャとプロシージャの間にはなにも入らず，区切りが見にくいものになってしまいます．プロシージャの終りの「end」と次のプロシージャの「to」の間になにも入らないのです．

そこでプロシージャとプロシージャの間を1行あけてプリントアウトするLOGOPRNTというコマンドを作ったわけです.

実行する際カーソルパドルを使ってコマンドを選択するときはPRINTと同じですが，直接キーボードから入力するとき，ファイル名に拡張子はつけなくても自動的に.LOGを補ってくれます.

ソースプログラム

List 8-7

```
                     10 ';================================================
                     20 ';                  Print out Logo file
                     30 ';================================================
                     40 ';
00F0 41319800        50 '          org     0f0h
00F4 0F00F000
00F8 58000001        60 '          entry   x
00FC FF00
00FE 0000            70 '          start
                     80 ';
     0005            90 'bdos      equ     0005h
     005C           100 'fcb       equ     005ch
                    110 ';
                    120 ';------------------------------------------------
                    130 ';                        open file
                    140 ';------------------------------------------------
                    150 ';
0100 216500         160 'x:        hl=65h  ;5ch+9 first letter of extension
0103 7E             170 '          a=(hl)
0104 FE20           180 '          a-32
0106 2008           190 '          ->lpextexi,~0
0108 364C           200 '          (hl)=4ch        ;L
010A 23             210 '          hl+
010B 364F           220 '          (hl)=4fh        ;O
010D 23             230 '          hl+
010E 3647           240 '          (hl)=47h        ;G
0110 115C00         250 'lpextexi:de=fcb   ;extension exists
0113 0E0F           260 '          c=15    ;file open
0115 CD0500         270 '          ]bdos
0118 3C             280 '          a+
0119 2021           290 '          ->lpcont1,~0
                    300 ';
                    310 ';--- file not found ---
                    320 ';
011B 2E0B           330 '          l=11    ;xcor
011D 260C           340 '          h=12    ;ycor
011F 1600           350 '          d=0
0121 0E15           360 '          c=21    ;move cursor
0123 CF             370 '          ]'08
0124 112B01         380 '          de=lpmessa
0127 0E09           390 '          c=9     ;CRT-out message
0129 CF             400 '          ]'08
012A C9             410 '          [
012B 46696C65       420 'lpmessa: db       'File not found!!',0
012F 206E6F74
0133 20666F75
0137 6E642121
013B 00
                    430 ';
                    440 ';------------------------------------------------
                    450 ';                             main
                    460 ';------------------------------------------------
                    470 ';
013C 217401         480 'lpcont1: hl=lptable
013F 0E05           490 '          c=5
```

```
0050  0141 E5          500 ´lpnextsc:#=hl
0051  0142 C5          510 ´         #=bc
0052  0143 115C00      520 ´         de=fcb
0053  0146 0E14        530 ´         c=20      ;read file
0054  0148 CD0500      540 ´         Jbdos
0055  014B 3C          550 ´         a+
0056  014C 2001        560 ´         ->lpnotend,~0
0057  014E C7          570 ´lpend:   J´00
0058                   580 ´;
0059  014F C1          590 ´lpnotend:bc=#
0060  0150 E1          600 ´         hl=#
0061                   610 ´;
0062  0151 118000      620 ´         de=80h
0063  0154 0680        630 ´         b=80h
0064  0156 1A          640 ´lploop1: a=(de)
0065  0157 FE1A        650 ´         a-1ah
0066  0159 28F3        660 ´         ->lpend,0
0067  015B CD7901      670 ´         Jprinter
0068  015E BE          680 ´         a-(hl) ;search LF,'end',CR
0069  015F 2009        690 ´         ->lpcont2,~0
0070  0161 23          700 ´         hl+
0071  0162 0D          710 ´         c-
0072  0163 200A        720 ´         ->lpcont3,~0
0073  0165 3E0A        730 ´         a=10      ;found LF,'end',CR
0074  0167 CD7901      740 ´         Jprinter
0075  016A 217401      750 ´lpcont2: hl=lptable
0076  016D 0E05        760 ´         c=5
0077  016F 13          770 ´lpcont3: de+
0078  0170 10E4        780 ´         ~>lploop1
0079  0172 18CD        790 ´         ->lpnextsc
0080                   800 ´;
0081  0174 0A656E64    810 ´lptable: db        10,'end',13
      0178 0D
0082                   820 ´;
0083                   830 ´;-----------------------------------------------
0084                   840 ´;                  char-out to printer
0085                   850 ´;-----------------------------------------------
0086                   860 ´;
0087  0179 F5          870 ´printer: #=af
0088  017A C5          880 ´         #=bc
0089  017B D5          890 ´         #=de
0090  017C E5          900 ´         #=hl
0091  017D 5F          910 ´         e=a
0092  017E 0E05        920 ´         c=5        ;lprint
0093  0180 CD0500      930 ´         Jbdos
0094  0183 E1          940 ´         hl=#
0095  0184 D1          950 ´         de=#
0096  0185 C1          960 ´         bc=#
0097  0186 F1          970 ´         af=#
0098  0187 C9          980 ´         [
0099                   990 ´;
0100  0188            1000 ´         end

0005 BDOS      005C FCB       013C LPCONT1   016A LPCONT2   016F LPCONT3
014E LPEND     0110 LPEXTEXI  0156 LPLOOP1   012B LPMESSA   0141 LPNEXTSC
014F LPNOTEND  0174 LPTABLE   0179 PRINTER   0100 X
```

リスト8－7にソースプログラムをアセンブルした際のリスティングファイル（アセンブルのときオプションとして／Lを付けた結果できた，拡張子が.LSTとなっているファイル）を示します．

50～160 ：すでに気付いている方も多いと思いますが，777 Assemblerでアセンブルしたものは，最初の部分に何かくっついていてorg 100hとしてもリンクパッケージ本体は100hから始まりません．そこでorg 0f0hとし，entry xとエントリのラベルを1文字とすることで，プログラムの本体が100h番地から始まるようにしています．

100 ：UIFが書き込んでくれたファイル名をそのまま使うため，FCBを005CHからにします．

160～240：入力されたファイル名に拡張子が付いていないときは，LOGを補います．

330～420：ファイルが見つからなかったとき，画面の真中にFile not found！！というメッセージを出力する．トランジェントコマンドに制御が移ったとき，画面は40桁になっています．

RET（ANN表記では［ ）でシステムメニューに戻っていますから，「Hit any key！！！」というメッセージが出力されます．

570 ：読むセクタがなくなって終了するときは］'00でシステムメニューに戻ります．このときは「Hit any key ！！！」のメッセージは出力されません．

620 ：DMAエリアは変更していませんから0080H番地からになっています．

実行形式ファイルの作成

ソースをアスキーセーブしたらシステムメニューに戻ってアセンブルしますが，このときは必ず／Aを付けて，アブソリュート・リンクパッケージを作るようにしてください．／Aを付けないと，orgは無視されて0100番地からのパッケージができてしまいます．

```
ASSEM LOGOPRNT/A ↵
```

こうしてできたアブソリュート・リンクパッケージを「DEBUG ××××××××．PAC」と入力してDebuggerでロードします．Debuggerに入るときファイルを指定すると，Rコマンドでファイルを読み込んだときと同様になります．

```
DEBUG  LOGOPRNT.PAC ↵
```

作成したリンクパッケージにはもともと00F0H～00FFHの余分なものが付いていますし，リンクパッケージファイルにはさらに「スタンダードラベル」が17バイト「ネクストモジュールフラグ」が2バイトの合計13hバイト余計なものが付いています．そこで先頭から都合23hバイトはいらないわけですから，「M123，××××，100」として除きます．

```
M123, 1FF, 100 ↵
```

0123からどこまでを移動させるかは，基本的には，

リスティングファイルに出ているプログラムの最終番地＋23h

までですが，余裕をみる分には問題ありません．

「Y××××××××.EXE」でファイル名を設定して「S100，××××」でディスクにセーブして完成です．

YLOGOPRNT.EXE ⏎
S100, 1FF ⏎

8-4 割込み

さきにも触れたようにSMC-777ではモード1の割込み（割込みがかかったら単に38H番地がコールされる）を使います．割込み源としては，60Hz割込み（BASICのSOUND命令はこれを使っている），キーボードが押されたときの割込みがあります．

(1) SMC-777における割込み

Sony Filerの下ではRAMは図8-5のようになっていて，ROMの0038H番地は，

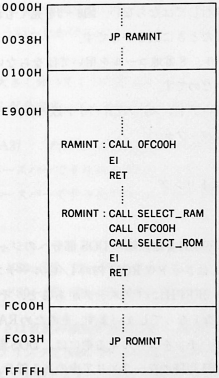


図8-5 割り込み関係のアドレスの様子

```
JP 0FC03H
```

になっています．シャドウROM内からでも割込み処理が可能になっているわけです．Sony Filerがコールドスタート，ウォームスタートするとFC00H番地には，

```
EI
RET
```

が書きこまれ，またBASIC中ではFC00H番地はDOS内の，

```
EI
RET
```

という部分へのジャンプ命令（C3 ×× ××となっているということ）になっています．BASIC中ではSOUNDコマンドを実行しない限りFC00H～FC02Hを書き換えないので，リンクパッケージで割込みを使うことが可能です．リンクパッケージをリンクして，FC01H，FC02H番地をそのリンクパッケージのなかの割込み処理のエントリに書き換えればよいわけです（FC00HはもともとC3Hになっているから)．ただし，割り込みプログラムを書くには以下のような注意が必要です．

・処理終了時にはEI命令を実行してはならない．図8－5を見てもわかるようにシャドウROM内から割り込みを受け付けたときに困るからです．
・処理の内部では5番地コール，8番地コールを用いてはならない．これはDOS部分がリエントラントになっていないためです．
・処理の手前（0000番地に近いアドレス）の257バイト以内に，
　①行入力，エディット用バッファ
　②FCB
　③Function key定義のストリング

　があってはならない．

0005H番地，0008H番地にはE900H番地以降のDOS部分へのジャンプ命令が書いてあります．そして，このDOS部分からさらにシャドウROM内のI／Oルーチンがコールされることが多くあります．RAM上の0000H～3FFFHにパラメータがあると，それはシャドウROMにバンクが切り換わったときには見えなくなってしまいます．そのためRAM上のパラメータを使用するようなシャドウROM内のルーチンをコールする前には，DOS部分で一度，そのパラメータの始めから257バイトを，FC00H以降のワークエリア中のある部分と，SWAPさせるのです．そういったシャドウROM内のルーチンの実行中に割込みがかかったときのことを考慮して，こういった制限が付けられるのです．

(2) 割込みを利用したサンプル・プログラム

割込みを使ったリングパッケージの実例をリスト8-8に示します．これはタイトルを見てもわかるようにテキスト表示のハードコピーを取るためのもので，CTRL +スペースバーでハードコピーが始まるようになっています．このプログラムはキーが押されたときにかかる割込みを利用しています．割込みがかかると，420行のintroutのところへ飛び，押されたキーを調べて，それがCTRL +スペースバーだったらハードコピーを始めるわけです．

しかし，どのキーが押されたか調べるにあたってはちょっとした問題があります．7章で述べたとおり，1AHポートを読めば押されたキーに対応するキャラクターコードはわかります．しかし，1AHポートが一度読まれると，キーが押されたことが認識されたとして，1AHポートからはもう同じデータは読み出せなくなります．ですから，1AHポートを読んで，どのキーが押されたかを判断していたのでは，事実上キー入力が全くできなくなってしまいます．キーが押されるたびに割込みがかかり，割込み処理のルーチン内でキーデータを読んでしまって，通常のルーチンがキーデータを読めないのです．

ところで，カーソルパドルとスペースバーには通常のキーとしての機能と，ジョイスティックとしての機能があります．これらのキーには，それぞれ2つずつスイッチが付いていて，ひとつがキーボードエンコーダーに，もうひとつがジョイスティックとしてCPUにつながっているのです．スペースバーが押されたことを知るには1AHポートを読まなくても，ジョイスティックのポートを読めばよいのです．こういったわけで，ハードコピーを始めるのにCTRL +スペースバーを使ったのです．

このリンクパッケージは大きく分けて2つの部分から成ります．モードを設定する部分と，実際に割込みに対する処理をする部分です．モード設定は，以下のようなコマンドで行います．

_COPYMODE(A1, A2, A3)　　ただしA1, A2, A3は0か1でA2, A3は省略可能

A1　0：CTRL +スペースバーでテキストのハードコピーを行わない
　　1：CTRL +スペースバーでテキストのハードコピーを行う
A2　0：横80桁
　　1：横40桁
A3　0：キャラクタコード80～9F，E0～FFをひらがなとして出力する
　　1：キャラクタコード80～9F，E0～FFをグラフィック文字として出力する

A2とA3は，省略されたときは直前の設定が有効になり，最初の設定はすべて0となります．ひらがなが出せるのは純正プリンタとNEC PC-8821またはその相当品だけです．8番地コールで画面が40桁か80桁かは知ることができるのですが，前節の注意にあったとおり，割込み処理ルーチンでは5番地・8番地コールが使えませんのでコマンドで認定するようになっています．また同様にプリンタに出力するルーチンもこのプログラムのなかにあるものを使っています．その

ほか，このプログラムの仕様として，ESC を押せばハードコピーは必ず中断されるようになっています．

List 8-8

```
0001                      10 ';=============================================
0002                      20 ';  Text screen hard-copy triggerd by ctrl space
0003                      30 ';=============================================
0004                      40 ';
0005  0100 52310801       50 '          org      100h
      0104 16000001
0006  0108 434F5059       60 '          entry    copymode(byt,[byt],[byt])
      010C 4D4F4445
      0110 001A0116
      0114 01
0007  0115 00034343       70 '          start
      0119 00
0008                      80 ';
0009                      90 ';---------------------------------------------
0010                     100 ';                      Mode setting
0011                     110 ';---------------------------------------------
0012                     120 ';
0013  011A 7E            130 'copymode:a=(hl)
0014  011B B7            140 '          a=a|a
0015  011C 2003          150 '          ->cm2,~0
0016  011E D31B          160 '          p(1bh)=a
0017  0120 C9            170 '          [
0018                     180 ';
0019  0121 DD7E01        190 'cm2:      a=(ix+1)
0020  0124 FE40          200 '          a-40h
0021  0126 2804          210 '          ->cm3,0
0022  0128 1A            220 '          a=(de)
0023  0129 329B01        230 '          (cowidth)=a
0024  012C DD7E02        240 'cm3:      a=(ix+2)
0025  012F FE40          250 '          a-40h
0026  0131 2804          260 '          ->cm4,0
0027  0133 0A            270 '          a=(bc)
0028  0134 329C01        280 '          (cohiraf)=a
0029                     290 ';
0030  0137 F3            300 'cm4:      di
0031  0138 3E01          310 '          a=1
0032  013A D31B          320 '          p(1bh)=a
0033  013C 214401        330 '          hl=introut
0034  013F 2201FC        340 '          (0fc01h)=hl
0035  0142 FB            350 '          ei
0036  0143 C9            360 '          [
0037                     370 ';
0038                     380 ';---------------------------------------------
0039                     390 ';                        Key sense
0040                     400 ';---------------------------------------------
0041                     410 ';
0042  0144 F5            420 'introut: #=af
0043  0145 DB1B          430 '          a=p(1bh)          ;key board status
0044  0147 B7            440 '          a=a|a
0045  0148 FA4D01        450 '          =>int1,-          ;sense ctrl
0046  014B F1            460 '          af=#
0047  014C C9            470 '          [
0048  014D C5            480 'int1:     #=bc
0049  014E 015101        490 '          bc=151h
0050  0151 ED78          500 '          a=p(c)            ;sense 'shot'(=space bar)
0051  0153 E610          510 '          a=a&00010000b
0052  0155 2803          520 '          ->int2,0
0053  0157 C1            530 '          bc=#
0054  0158 F1            540 '          af=#
0055  0159 C9            550 '          [
```

```
                    560 ';
                    570 ';-----------------------------------------------
                    580 ';                        Main
                    590 ';-----------------------------------------------
                    600 ';
015A DB1A           610 'int2:    a=p(1ah)          ;key data
015C D5             620 'copy:    #=de
015D E5             630 '         #=hl
015E 08             640 '         af:af'
015F F5             650 '         #=af
0160 ED739D01       660 '         (spstack)=sp
0164 3A9B01         670 '         a=(cowidth)
0167 B7             680 '         a=a!a
0168 2005           690 '         ->co1,~0
016A 115000         700 '         de=0050h;00:nop,50h=80
016D 1803           710 '         ->co2
016F 112803         720 'co1:     de=0328h;03:inc bc,28h=40
0172 218101         730 'co2:     hl=co4
0175 72             740 '         (hl)=d
0176 CDD801         750 '         ]lpcrlf
0179 010000         760 '         bc=0
                    770 ';
017C 53             780 'coloop2: d=e
017D CD9F01         790 'coloop1: ]getprint
0180 03             800 '         bc+
0181 00             810 'co4:     ds      1         ;written 'nop' or 'bc+'
0182 15             820 '         d-
0183 20F8           830 '         ->coloop1,~0
0185 CDD801         840 '         ]lpcrlf
0188 21CF07         850 '         hl=1999
018B B7             860 '         a=a!a             ;clear carry
018C ED42           870 '         hl=hl-bc@
018E 30EC           880 '         ->coloop2,~@
0190 ED7B9D01       890 'co5:     sp=(spstack)
0194 F1             900 '         af=#
0195 08             910 '         af:af'
0196 E1             920 '         hl=#
0197 D1             930 '         de=#
0198 C1             940 '         bc=#
0199 F1             950 '         af=#
019A C9             960 '         [
019B 00             970 'cowidth: ds      1
019C 00             980 'cohiraf: ds      1
019D 0000           990 'spstack: ds      2
                   1000 ';
                   1010 ';-----------------------------------------------
                   1020 ';     Get a charactor from V-RAM and print out
                   1030 ';-----------------------------------------------
                   1040 ';
                   1050 ';--- get char ---
                   1060 ';
019F 60            1070 'getprint:h=b
01A0 41            1080 '         b=c
01A1 4C            1090 '         c=h
01A2 ED78          1100 '         a=p(c)
01A4 60            1110 '         h=b
01A5 41            1120 '         b=c
01A6 4C            1130 '         c=h
                   1140 ';
01A7 FE21          1150 '         a-21h
01A9 3004          1160 '         ->gp1,~@
01AB 3E20          1170 'gp2:     a=20h
01AD 1839          1180 '         ->lprint
01AF FE7F          1190 'gp1:     a-7fh
01B1 28F8          1200 '         ->gp2,0
01B3 3833          1210 '         ->lprint,@
01B5 67            1220 '         h=a
01B6 3A9C01        1230 '         a=(cohiraf)
01B9 B7            1240 '         a=a!a
01BA 7C            1250 '         a=h
01BB 202B          1260 '         ->lprint,~0
01BD FEA0          1270 '         a-0a0h
```

```
01BF 3807        1280 '            ->gphira1,@
01C1 FEE0        1290 '            a-0e0h
01C3 3823        1300 '            ->lprint,@
01C5 E6DF        1310 '            a=a&11011111b
01C7 21          1320 '            db     21h
01C8 F620        1330 'gphira1:  a=a!00100000b
01CA 08          1340 '            af:af'
01CB 3E26        1350 '            a=26h;'&'
01CD CDE101      1360 '            ]lprintes
01D0 08          1370 '            af:af'
01D1 CDE801      1380 '            ]lprint
01D4 3E24        1390 '            a=24h;'$'
01D6 1809        1400 '            ->lprintes
                 1410 ';
                 1420 ';-----------------------------------------------
                 1430 ';           Subroutines to output to printer
                 1440 ';-----------------------------------------------
                 1450 ';
01D8 260D        1460 'lpcrlf:    h=13
01DA CDE901      1470 '            ]lploop
01DD 260A        1480 '            h=10
01DF 1808        1490 '            ->lploop
01E1 6F          1500 'lprintes:l=a
01E2 261B        1510 '            h=1bh
01E4 CDE901      1520 '            ]lploop
01E7 7D          1530 '            a=l
01E8 67          1540 'lprint:    h=a
                 1550 ';
                 1560 ';--- Sense ESC key ---
                 1570 ';
01E9 DB1B        1580 'lploop:   a=p(1bh)
01EB 1F          1590 '            >(.@)
01EC 3008        1600 '            ->lp1,~@
01EE DB1A        1610 '            a=p(1ah)
01F0 FE1B        1620 '            a-1bh
01F2 2002        1630 '            ->lp1,~0
                 1640 ';
01F4 189A        1650 '            ->co5
01F6 DB1D        1660 'lp1:       a=p(1dh)
01F8 E610        1670 '            a=a&10h
01FA 20ED        1680 '            ->lploop,~0
01FC 7C          1690 '            a=h
01FD D322        1700 '            p(22h)=a
01FF 3E16        1710 '            a=16h
0201 D31C        1720 '            p(1ch)=a
0203 3E06        1730 '            a=06h
0205 D31C        1740 '            p(1ch)=a
0207 C9          1750 '            [
                 1760 ';
0208             1770 '            end

0121 CM2        012C CM3        0137 CM4        016F CO1        0172 CO2
0181 CO4        0190 CO5        019C COHIRAF    017D COLOOP1    017C COLOOP2
015C COPY       011A COPYMODE   019B COWIDTH    019F GETPRINT   01AF GP1
01AB GP2        01C8 GPHIRA1    014D INT1       015A INT2       0144 INTROUT
01F6 LP1        01D8 LPCRLF     01E9 LPLOOP     01E8 LPRINT     01E1 LPRINTES
019D SPSTACK
```

APPENDIX

1 Logo

(1) エスケープ文字について

Logoでは

スペース ( ) ＊＋－／ ＜＝＞［ ¥ ］；

の14文字は特別な意味を持っています．通常これらの文字はその他の文字のようにプロシージャ名や変数名に使うことはできません．例えばab＊cdという変数に10を代入しようとして，

? make "ab＊cd 10 ↵

と入力しても，Logoは，

＊ doesn't like ab as input

とエラーメッセージを返してきます．つまり＊を乗算の演算記号と解釈してしまうわけです．

そこでLogoには，「この文字の次にある文字は，特別な意味は持っていない」ということを示す文字として¥が使われます．こういった働きをする文字をエスケープ文字といいます．たとえばab＊cdに10を代入したいときは，

? make "ab¥＊cd 10 ↵

とすればよいのです．また¥を変数名などに含めたいときは，

¥¥

と2つ重ねます．例を挙げますと，

```
? make "¥¥ 27 [↵]
? "¥¥ [↵]
¥
? :¥¥ [↵]
27
```

ということになります．

（2）変数名

本文中で何度も出てきたように，変数に値を代入するときは，

make "a 100

のようにします．これを本文中では，

make "〈変数名〉〈値〉

という書式で説明しました．しかし，マニュアル等を見ると make の書式は，

make 〈変数名〉〈値〉

となっていて，" のマークはありません．これはどうしてでしょうか？

Logo の命令の書式は一般に，

〈命令語〉〈引数１〉〈引数２〉……

であることは，本文でも触れたとおりです．

この引数には一般に，" のマークの付いた定数の型以外に，: のマークの付いた変数の型や，値を返すプリミティブを置くこともできるのです．ですから make の場合，

make :a 100

という書き方も可能なのです．

今までは，make の１つ目の引数として " マークの付いた定数の型だけを置いていたので，本文中では "〈変数名〉と表現していましたが，本来なら，

make 〈変数名〉〈値〉

とする方が正しいのです．

次に，make の１つ目の引数に，: のマークの付いた，変数の型を使った場合の働きを説明しま

す．例を見てください．

?make "セイコ "サンスウ ⏎

?make :セイコ 70 ⏎

? :セイコ ⏎

サンスウ

? :サンスウ ⏎

70

この例でもわかるように，:セイコをmakeの1つ目の引数にした場合は，"セイコという変数名の指す内容であるサンスウを求め，

?make "サンスウ 70 ⏎

としたときと同じ働きをするのです．

また，makeとは逆の働きを持つプリミティブに，thingというものがあり，

?make "アキナ "シャカイ

?make :アキナ 80

?thing "アキナ

シャカイ

?thing :アキナ

80

となります．

ここで説明したことは，初心者の方にはわかりにくいかもしれません．しかし，これらの機能を使うと間接参照などが行えるようになり，プログラミングに幅を持たせることができます．興味のある方は，次のプログラムとその実行結果を参考にして，自分で工夫してみてください．

List A-1

```
to build
make "カモク [コクゴ" サンスウ リカ シャカイ]
make "セイコ "カモク
make "アキナ "カモク
make "キョウコ "カモク
make "イヨ "カモク
make "man [セイコ アキナ キョウコ イヨ]
make "コクゴ" [100 20 40 70]
make "サンスウ [20 30 40 90]
make "リカ [0 16 32 87]
make "シャカイ [99 77 34 50]
end
```

```
to do
make "i 0
repeat count :man [
   make "i (+ :i 1)
   make "j (item :i :man)
   pr [] type :j
   make "k (thing :j)
   pr :k
   make "l 0
   repeat count (thing :k) [
       make "l (+ :l 1)
       make "m (item :l (thing :k))
       type :m
       type (item :i (thing :m))]]
end
```

```
?build
?do

セイコカモク
コクゴ"100サンスウ20リカ0シャカイ99
アキナカモク
コクゴ"20サンスウ30リカ16シャカイ77
キョウコカモク
コクゴ"40サンスウ40リカ32シャカイ34
イヨカモク
コクゴ"70サンスウ90リカ87シャカイ50
?
```

リストA-1の実行結果

(3) 演算子

Logoでは引数に算術式が許されています．演算子には＋－＊／があり，それぞれ加減乗除に対応しています．当然，乗除算が優先です．

ところで，＋－＊／はプリミティブとしても働き，

? + 10 20 ↵

30

のようになります．プリミティブとしての＋＊は引数を3つ以上とれるので，便利なことがあります．たとえば，

```
? (＊1 2 3 4 5 6 7 8 9 10) ↵
362880
? (＋1 2 3 4 5 6 7 8 9 10) ↵
55
```

となります.

(4) catchとthrow

Dr. Logoでは，プログラムの流れを制御するプリミティブとしてcatchとthrowという独特な命令が用意されています．初めはちょっとわかりにくいプリミティブですが，慣れると便利ですのでよく勉強して活用してください．

BASICに慣れている方は，GOSUB～RETURNを思い出していただくとわかり易いかと思います（使い方は，ちょっと違いますけど…）．GOSUB～RETURN文では，GOSUB文で指定したサブルーチンに実行が移され，サブルーチン中のRETURNを認識すると，メインルーチンのGOSUB文の次から実行を始める，という流れが実行されます．Logoでは，GOSUBがcatchに，RETURNがthrowに当たります．簡単な構造としては，Logoがcatchを見つけると，catchの引数である命令リスト（他のプロシージャや命令等）を実行します．そして，この命令リストを実行中に，Logoがthrowを評価すると，throwの引数と同じ名前を持つcatchを探して，そこへ戻り，そのcatchの次から実行を再開します．GOSUB～RETURNと異なるのは，サブルーチン（他のプロシージャ）をコールするだけでなく，catchの引数の中にthrowが存在してもよいこと．それに，catchで呼んだプロシージャが呼んだプロシージャからthrowで2レベル戻すこともできます．たとえば，

```
catch "teru [if :n > :m [throw "teru] [Jun] ]

to a
catch "b [c]
end

to c
d
end

to d
```

```
throw "b
end
```

といった構成が可能なわけです．つまり１つ目では：n＞：m が TRUE であれば，次の命令に制御が移り，FALSE ならば Jun というプロシージャをコールしろという命令になるわけです．また２つ目の例ではプロシージャ a が catch の中で c を呼び，この c が d を呼びます．そしてプロシージャ d の中の throw "b で c をすっ飛ばして直接プロシージャ a の catch の次へ行ってしまいます．

catchとthrowには，この他にも便利な使い方があります．

まず，catch の名前に error を使うと，エラートラップとして利用できます．プロシージャの実行中にエラーが発生すると自動的に catch "error 文の次に実行が移ります．この場合，対応する throw "error を書く必要はなく，Logo が自動的に throw "error を補って catch の次に実行を移します．

また，catch の引数に TRUE を用いると，throw の引数に用いられる名前に関係なく，catch "TRUE の次に実行を移すことができます．

throw では，引数に TOPLEVEL を指定することによりプロシージャの実行を終了させることもできます．通常，プロシージャを実行するとき，ほかのプロシージャをどんどんコールしてゆきます．そして，下位のレベルのプロシージャで実行を終了したいとき，stop を用いるとプロシージャは実行を終了し，そのプロシージャをコールしたプロシージャに実行を移してしまいます．このため，下位のレベルで全プロシージャの実行を終了したいときに，この throw "TOPLEVEL を使用します．

このように，catchとthrowは，プログラムの流れを制御したり，エラートラップをしたり，プロシージャの実行を終了したり，様々な働きをします．

（５）システムプロパティとシステム変数

Logo の処理系自身，データはすべてプロパティで管理しています．プロシージャの内容や変数などすべてプロパティとして保存されているのです．こういった Logo の処理系自身が作るプロパティのプロパティ名を表A－1に示します．ユーザーが作るプロパティはこれ以外のプロパティ名を使わなければなりません．

また，その変数の値が直接 Logo の動作にかかわってくる，いわばシステム変数と呼べるものが２つあります．

ひとつめは REDEFP で，これは通常未定義状態ですが，

```
make "REDEFP "TRUE
```

を実行すると，プロシージャ名にプリミティブの名前を使えるようになります．ただし一度プロシージャ名としてプリミティブを使いますと，そのプリミティブはなくなってしまいますので注意が必要です．

もうひとつはERRACTというもので，これをTRUEにするとエラーが起こったとき，エラーメッセージを表示せず，ポーズ状態になります．このときエラーメッセージはerrorというプリミティブが返してくれます．

表A-1 システムプロパティ一覧

プロパティ名	そのプロパティを作成するプリミティブ	プロパティの値の意味
.APV (application value)	make, ed, edall	グローバルな変数の値
.DEF (definition)	to, ed, edall, define	プロシージャの定義内容のリスト
.REM (remarks)		プロシージャの注釈文を示す
.FMT (format)	to, ed, edall	プログラムとしての1行を画面上の数行に分け，インデントした形式で登録した場合の形式情報
.ENL (end of line)	to, ed, edall	画面上の1行の終りに注釈文が付加されていることを示す
.SPC (space)		画面上の表示形式としてスペースが挿入されている数を示す
.FUN (function)		現在評価されている関数を示す．一時的なものなので，ユーザーは見ることはできない
.CAT (catch)	catch, throw	catchを登録するプロパティ．プロシージャ実行中に一時的に作られるものなので，ユーザーは見ることはできない
.PAU (pause)		ポーズ中，一時的に作られるプロパティ
.PRM (primitive)		プリミティブの処理ルーチンの先頭番地を示すプロパティ

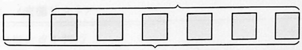
(6) プリミティブ一覧表

Dr. Logo のプリミティブの書式とその意味をABC順に並べますので，プログラム作りの際の参考にしてください．なお各プリミティブにはそれぞれの働きに応じて分類記号を付けましたので，検索の際に役立ててください．

プリミティブの分類

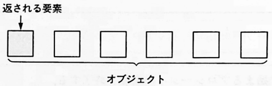
TEXT	テキストスクリーンに関するプリミティブ
GRAP	グラフィックスクリーンに関するプリミティブ
TURTL	タートル制御に関するプリミティブ
PALET	カラーパレットに関するプリミティブ
CALC	演算機能に関するプリミティブ
W&L	ワードとリストの処理に関するプリミティブ
DEF	プロシージャの定義と修正に関するプリミティブ
NUM	変数に関するプリミティブ
IF	条件判断に関するプリミティブ
PROC	プロシージャの実行に関するプリミティブ
WORK	ワークスペース管理に関するプリミティブ
DEBUG	デバッグに関するプリミティブ
PROP	プロパティ管理に関するプリミティブ
KEY	キーボードと接続機器に関するプリミティブ
FILE	ファイルとディスクに関するプリミティブ
DATA	データの直接入力に関するプリミティブ

A	・and 書式　and 〈判定文〉〈判定文〉…… 判定文の結果がすべて真の場合 TRUE を返し，それ以外は FALSE を返すプリミティブ．引数が2つ以外のときは全体を(　)でくくる．	IF
	・arctan 書式　arctan 〈数値〉 逆三角関数アークタンジェント $\tan^{-1}(x)$ の値を返す．	CALC

A	・ascii 書式　　ascii 〈ワード〉 与えられたワードの先頭文字のアスキーコードを返す．	W&L
B	・back (bk) 書式　　bk 〈ステップ数〉 タートルの向きと反対方向に，与えられたステップ数分だけ後退する．	TURTL
	・butfirst (bf) 書式　　bf 〈オブジェクト〉 与えられたオブジェクトの先頭の要素を取り除き，残りの要素を返す． 	W&L
	・buttonp 書式　　buttonp 〈パドル番号〉 スペースバーまたはジョイスティック1（パドル番号＝0），ジョイスティック2（パドル番号＝1）が押されていればTRUE，押されていなければFALSEを返す．	KEY
	・bye 書式　　bye Logoを終了してSony Filerのシステムメニューに戻る．	
C	・catch 書式　　catch 〈名前〉〈命令リスト〉 命令リストを実行中，同じ名前を引数に持つthrowを見つけると再びcatchに戻り，catchの次から実行する．APPENDIX(4)参照．	PROC
	・changef 書式　　changef 〈新ファイル名〉〈旧ファイル名〉 ドライブ中のディスクのLogoファイルのファイル名を変更する．	FILE

C	・char 書式　char　〈数値〉	W&L
	与えられた数値をアスキーコードとする文字を返す.	
	・clean 書式　clean	GRAP
	タートルの状態を変えず，グラフィック表示だけを消す.	
	・clearscreen（cs） 書式　cs	TURTL
	タートルをホーム位置に戻してグラフィック表示を消す.	
	・cleartext（ct） 書式　ct	TEXT
	テキスト表示を消す.	
	・co 書式　co　または　co　〈オブジェクト〉	PROC
	pause で中断されたプロシージャの実行を再開する.	
	・copyoff 書式　copyoff	KEY
	copyon の状態を解除して，プリンタへの出力を行わなくする.	
	・copyon 書式　copyon	KEY
	これを実行すると，それ以後画面上に出力される文字をプリンタにも出力する. copyoff が実行されるまでプリンタ出力状態が継続する.	
	・cos 書式　cos	CALC
	三角関数コサイン cos(x)の値を返す．角度は度（degree）で与える.	
	・count 書式　count　〈オブジェクト〉	W&L
	与えられたオブジェクトの要素数を返す.	

D	・cursor 書式　cursor 現在のカーソルの位置を座標値リストで返す.	TEXT
	・defaultd 書式　defaultd 現在の登録ドライブ名を返す.	FILE
	・define 書式　define 〈プロシージャ名〉〈定義リスト〉 定義リストを指定された名前でプロシージャとして登録する.	DEF
	・dot 書式　dot 〈座標リスト〉 指定された座標に点を打つ.	GRAP
E	・edall 書式　edall ワークスペース内のすべてのプロシージャとグローバル変数をエディットモードに移行する.	DEF
	・edit (ed) 書式　ed 〈名前〉または ed 〈名前リスト〉 指定の名前を持つプロシージャとグローバル変数をエディットモードに移行する.	DEF
	・emptyp 書式　emptyp 〈オブジェクト〉 与えられたオブジェクトの要素数が0のときTRUEを返し,それ以外はFALSEを返す.	W&L
	・end 書式　end to プロシージャ名で始まるプロシージャの定義を終了する.	DEF
	・equalp (=) 書式　= 〈オブジェクト〉〈オブジェクト〉 与えられたオブジェクト同士が等しければTRUE, そうでなければFALSEを返す.	IF

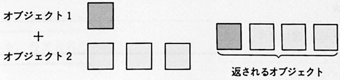
E	・erall 書式　erall ワークスペース内のすべてのプロシージャとグローバル変数を削除する．	WORK
	・erase (er) 書式　er 〈プロシージャ名〉または　er 〈プロシージャ名リスト〉 指定のプロシージャを削除する．	WORK
	・erf 書式　erf 〈ファイル名〉 ディスク上の，ファイル名で指定したファイルを削除する．	FILE
	・ern 書式　ern 〈変数名〉または　ern 〈変数名リスト〉 指定した変数名を持つグローバル変数を削除する．	WORK
	・error 書式　error 最後に起こったエラーの内容をリストで返す．システム変数 ERRACT を TRUE にしておくと，エラーが起こったとき，エラーメッセージを表示しなくなる．このときに有効になる．	DEBUG
F	・fence 書式　fence タートルが画面の外に出ないように画面のまわりに柵を設定する．	GRAP
	・first 書式　first 〈オブジェクト〉 与えられたオブジェクト中の最初にある要素を返す． 	W&L
	・forward (fd) 書式　fd 〈ステップ数〉 タートルを与えられたステップ数だけ前進させる．	TURTL

F

・fput [W&L]

書式　fput 〈オブジェクト1〉〈オブジェクト2〉

与えられたオブジェクト1をオブジェクト2の最初の要素として合成して返す．



・fullscreen（F3） [GRAP]

書式　fullscreen

画面全体をグラフィック表示だけに使うように設定する．

G

・getfs [FILE]

書式　getfs 〈ファイル指定〉

ディスク上に存在するLogoのファイル名をリストで返す．

・glist [PROP]

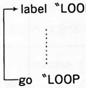
書式　glist 〈プロパティ名〉

与えられたプロパティ名の内容を持つ名前のリストを返す．

・go [PROC]

書式　go 〈名前〉

同じ名前を引数に持つlabelに実行を移す．



・gprop [PROP]

書式　gprop 〈名前〉〈プロパティ名〉

指定された名前の指定されたプロパティの内容を返す．

H

・heading [TURTL]

書式　heading

現在タートルが向いている方向を絶対角度（基準回転軸から右まわりに何度か）で返す．

H

・hideturtle（ht）　TURTL

書式　ht

タートルを画面に表示させないようにする．描画が速くなる．

・home　TURTL

書式　home

タートルをホーム位置に戻す．

I

・if　IF

書式　if 〈判定文〉〈命令リスト 1〉〈命令リスト 2〉

判定文の値が TRUE ならば命令リスト 1，そうでなければ命令リスト 2 を実行する．命令リスト 2 を省略すると値が FALSE の場合，次に実行が移る．

・.in　DATA

書式　.in 〈ポート番号〉

指定ポートからデータを入力し，その値を返す．

・int　CALC

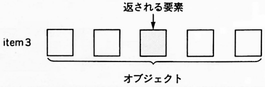
書式　int 〈数値〉

与えられた数値の整数部分を返す．
int　1234.56→1234

・item　W&L

書式　item 〈要素番号〉〈オブジェクト〉

指定されたオブジェクトから，指定された順番目にある要素を 1 つ返す．



K

・keyp　KEY

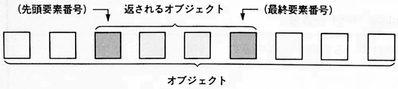
書式　keyp

キーボードが押されていれば TRUE，そうでなければ FALSE を返す．

L	・label 書式　label 〈名前〉 go により実行を移す位置を示す.	PROC
	・left (lt) 書式　lt 〈角度〉 タートルを，与えられた角度だけ左に回転させる.	TURTL
	・list 書式　list 〈オブジェクト〉〈オブジェクト〉…… 与えられたオブジェクトを要素とするリストを返す．引数が2つ以外の場合は，全体を( )でくくる.	W&L
	・listp 書式　listp 〈オブジェクト〉 与えられたオブジェクトがリスト形式であればTRUEを，それ以外ならFALSEを返す.	W&L
	・load 書式　load 〈ファイル名〉 ディスク上から指定のファイルを読込む.	FILE
	・local 書式　local 〈変数名〉〈変数名〉…… 指定の変数名を，それが宣言されたプロシージャ及びそこから呼ばれる下位レベルのプロシージャのなかでのみ有効な変数として設定する．引数が2つ以上のときは全体を( )でくくる.	NUM
	・lowercase (lc) 書式　lc 〈ワード〉 与えられたワードを小文字に変換する.	W&L
M	・make 書式　make 〈変数名〉〈値〉 指定された変数名の指す変数に値を代入する.	NUM
	・memberp 書式　memberp 〈オブジェクト1〉〈オブジェクト2〉 与えられたオブジェクト1が，オブジェクト2を構成する要素であればTRUE，そうでなければFALSEを返す.	W&L

N	・namep 書式　namep 〈名前〉 指定した名前のグローバル変数が存在していればTRUE，そうでなければFALSEを返す．	NUM
	・nodes 書式　nodes ワークスペース内の未使用ノード数を返す．	WORK
	・noformat 書式　noformat 形式プロパティを削除する．	WORK
	・not 書式　not 〈判定文〉 判定文の値がTRUEならばFALSE，FALSEならばTRUEを返す．	IF
	・notrace 書式　notrace トレースモードを終了する．	DEBUG
	・nowatch 書式　nowatch 〈プロシージャ名〉または nowatch 〈プロシージャ名リスト〉 指定のプロシージャのステップ動作を終了する．引数を省略すると，すべてのプロシージャについてステップ動作を終了する．	DEBUG
	・numberp 書式　numberp 〈オブジェクト〉 与えられたオブジェクトが数値であればTRUEを，そうでなければFALSEを返す．	W&L
O	・or 書式　or 〈判定文〉〈判定文〉…… 与えられた判定文のうち，どれかひとつでもTRUEがあればTRUEを返す．すべてがFALSEの場合はFALSEを返す．引数が2つ以外のときは全体を( )でくくる．	IF
	・.out 書式　. out 〈ポート番号〉〈出力データ〉 指定ポートにデータを出力する．	DATA

O	・output (op) 書式　op 〈オブジェクト〉 プロシージャの実行を終了し，オブジェクトの値をプロシージャの結果として返す．	PROC
P	・paddle 書式　paddle 〈パドル番号〉 パドルが押されている方向を数値で返す．	KEY
	・palette 書式　palette 〈パレット番号〉 指定された色のカラー成分をリスト［R G B］で返す．	PALET
	・pause 書式　pause プロシージャの実行を中断し，キーボードからの入力を受ける．	PROC
	・pen 書式　pen 現在のペン情報を返す．	GRAP
	・pencolor (pc) 書式　pc 現在，ペン（＝タートル）が持っているカラーコードを返す．	GRAP
	・pendown (pd) 書式　pd タートルを，線を描く状態にする．	GRAP
	・penerase (pe) 書式　pe タートルが線を消しながら移動するようにする．	GRAP
	・penreverse (px) 書式　px すでに画面に表示されている色とペンの色のカラーコードの排他的論理和(XOR) をカラーコードとする線を描くようにする．	GRAP
	・penup (pu) 書式　pu タートルが線を描かずに移動するようにする．	GRAP

P	・piece 書式　piece 〈先頭要素番号〉〈最終要素番号〉〈オブジェクト〉	W&L
	与えられた先頭要素番号から最終要素番号までにあるすべての要素をオブジェクト中から抜き出して返す．	
		
	・plist 書式　plist 〈名前〉	PROP
	指定した名前に登録されているプロパティリストの全要素を返す．	
	・po 書式　po 〈名前〉または po 〈名前のリスト〉	WORK
	指定のプロシージャまたはグローバル変数の内容を表示する．	
	・poall 書式　poall	WORK
	ワークスペース内の全プロシージャ，グローバル変数の内容を表示する．	
	・pons 書式　pons	WORK
	ワークスペース内の全グローバル変数の内容を表示する．	
	・pops 書式　pops	WORK
	ワークスペース内の全プロシージャの内容を表示する．	
	・pos 書式　pos	TURTL
	現在タートルがいる位置の絶対座標値リストを返す．	
	・pots 書式　pots	WORK
	ワークスペース内の全プロシージャのタイトルだけを表示する．	

P	・pprop 書式　pprop 〈名前〉〈プロパティ名〉〈内容〉 与えられたプロパティ名と内容をリストとして，指定された名前のプロパティリストに登録する．	PROP
	・pps 書式　pps ワークスペース内の全ユーザー定義プロパティを表示する．	PROP
	・print (pr) 書式　pr 〈オブジェクト〉〈オブジェクト〉… オブジェクトを画面に表示し，表示が終ると改行する．引数が1つ以外のときは全体を(　)でくくる．	TEXT
Q	・quote 書式　quote 〈オブジェクト〉 オブジェクトが直接評価されるのを防ぐ．	W&L
	quotient 書式　quotient 〈被除数〉〈除数〉 割り算の答を整数値で返す．小数点以下は切り捨てられる．	CALC
R	・random 書式　random 〈数値〉 0以上指定値未満，または指定値より大きく0以下（マイナスの数値を指定した場合）の整数値の乱数を返す．	CALC
	・readchar (rc) 書式　rc キーボードから1文字の入力を受け，その値を返す．⏎は必要なく，画面へのエコーバックもしない．	KEY
	・readquote (rq) 書式　rq キーボードから，⏎までの文字入力を受け，その結果をワード形式で返す．画面へエコーバックする．	KEY
	・recycle 書式　recycle ワークスペース内のゴミ集めを強制的に実行する．	WORK

R	・remainder 書式　remainder　〈被除数〉〈除数〉 整数除算の余りを返す.	CALC
	・remprop 書式　remprop　〈名前〉〈プロパティ名〉 指定したプロパティを名前で指定したプロパティリストから削除する.	PROP
	・repeat 書式　repeat　〈数値〉〈命令リスト〉 命令リストを数値で指定した回数だけ繰り返し実行する.	PROC
	・rerandom 書式　rerandom random，shuffleで発生する乱数を，特定の乱数系列の先頭に戻す.	CALC
	・resetpalette 書式　resetpalette　〈パレット番号〉 指定したカラーパレットの色を初期状態に戻す．引数を省略した場合はすべてのパレットを初期化する.	PALET
	・right (rt) 書式　rt　〈ステップ数〉 指定した角度だけタートルの向きを右回りに回転させる.	TURTL
	・round 書式　round　〈数値〉 数値の小数点以下第一位を四捨五入した値を返す.	CALC
	・run 書式　run　〈命令リスト〉 与えられた命令リストを実行する.	PROC
S	・save 書式　save　〈ファイル名〉 ワークスペース内のデータをすべてディスクに書き込む.	FILE
	・scrunch 書式　scrunch ドットの縦横比を補正する状態であれば「1」を，補正しない状態であれば「0」を返す.	GRAP

S	・sentence (se) 書式　se〈オブジェクト〉〈オブジェクト〉…… 与えられたオブジェクトの一番外側の[　]を取ったものを要素とするリストを返す．引数が２つ以外のときは全体を(　)でくくる．	W&L
	・setalpha 書式　setalpha〈カラーコード〉 テキスト表示画面で使われる文字の色を設定する．	TEXT
	・setborder 書式　setborder　〈カラーコード〉 ボーダーエリアの色を設定する．	GRAP
	・setcursor 書式　setcursor〈座標値リスト〉 指定した座標にカーソルを移動する．	TEXT
	・setd 書式　setd　〈ドライブ名〉 登録ドライブを指定する．	FILE
	・setheading (seth) 書式　seth〈絶対角度〉 タートルを指定の絶対角度（基準回転軸から右まわりに何度か）の方向に向ける．	TURTL
	・setpalette 書式　setpalette〈カラーコード〉〈RGB成分リスト〉 カラーパレットの色を設定する．	PALET
	・setpc 書式　setpc　〈カラーコード〉 タートルの持つペンの色を設定する．	TURTL
	・setpen 書式　setpen　〈ペン情報リスト〉 タートルの持つペンの色とペンの状態を同時に設定する．	TURTL
	・setpos 書式　setpos　〈座標リスト〉 タートルを指定の絶対座標に移動する．	TURTL

S	・setscrunch 書式　setscrunch　1 or 0 引数が「1」のとき1ドットの縦横比を補正する状態に,「0」のときは補正しない状態にする.	GRAP
	・setsplit 書式　setsplit 〈テキスト表示行数〉 スプリット画面にしたとき,画面の下から何行分テキスト表示領域をとするかを指定する.	GRAP
	・setx 書式　setx 〈X座標〉 タートルを指定のX座標位置(絶対座標)まで水平に移動させる.	TURTL
	・sety 書式　sety 〈Y座標〉 タートルを指定のY座標位置(絶対座標)まで垂直に移動させる.	TURTL
	・show 書式　show 〈オブジェクト〉〈オブジェクト〉…… オブジェクトをそのまま画面に表示し,表示後改行する.オブジェクトがリストの場合,ブラケット([　])も表示される.引数が2つ以上のときは全体を(　)でくくる.	TEXT
	・shownp 書式　shownp タートルが画面に表示されている状態ならばTRUE,そうでなければFALSEを返す.	TURTL
	・showturtle (st) 書式　st タートルを画面に表示する状態にする.	TURTL
	・shuffle 書式　shuffle 〈リスト〉 与えられたリストの各要素の並び方をランダムに入れ替える.	W&L
	・sin 書式　sin 〈角度〉 三角関数サインの値を返す.角度は度(degree)で与える.	CALC

S	・splitscreen（F2） 書式　splitscreen グラフィック画面の上にテキスト画面を重ねて表示する状態にする．	GRAP
	・stop 書式　stop 実行しているプロシージャを終了し，そのプロシージャをコールしたプロシージャに実行を戻す．	PROC
T	・text 書式　text 〈プロシージャ名〉 指定の名前で登録されているプロシージャの定義プロパティを返す．	DEF
	・textscreen（F1） 書式　textscreen 画面全体をテキスト表示だけに使う状態にする．	TEXT
	・thing 書式　thing 〈変数名〉 与えられた変数名に登録されている変数の値を返す．	NUM
	・throw 書式　throw 〈名前〉 同じ名前を引数に持つ catch の次に実行を移す．APPENDIX 1（4）参照．	PROC
	・to 書式　to 〈名前〉 プロシージャの定義を開始する．endを入力すると定義を終了する．	DEF
	・tones 書式　tones 〈音符データリスト〉 音程データ（0～49）と音長データ（0～32767：n/1000秒）を組み合わせた音符データにより音楽を演奏する．	KEY
	・towards 書式　towards 〈座標値リスト〉 現在のタートル位置から見た指定座標の方向を絶対角度で表す．	TURTL
	・trace 書式　trace トレースモードに入り，プロシージャ名，レベル番号，変数値を表示しながら実行するようにする．	DEBUG

T	・type 書式　type 〈オブジェクト〉〈オブジェクト〉…… 与えられたオブジェクトを画面に表示する．表示が終了しても改行しない．引数がひとつ以外のときは全体を（　）でくくる．	TEXT
U	・uppercase (uc) 書式　uc 〈ワード〉 与えられたワードを大文字にして返す．	W&L
V	・.vin 書式　.vin 〈ビデオ RAM アドレス〉 ビデオ RAM の指定アドレスからデータを読取り，そのデータを返す．	DATA
	・.vout 書式　.vout 〈ビデオ RAM アドレス〉〈出力データ〉 ビデオ RAM の指定アドレスにデータを書き込む．	DATA
W	・wait 書式　wait 〈数値〉 プロシージャの実行を，約1/60秒の数値倍だけ停止する．	PROC
	・watch 書式　watch 〈プロシージャ名〉 または watch 〈プロシージャ名リスト〉 指定したプロシージャについてプロシージャのレベル番号，プロシージャ名，次に実行する命令の表示を一命令ごとに行い，キーボードからの命令入力を受け付けるステップ動作をする状態にする．引数を省略するとすべてのプロシージャについて行う．	DEBUG
	・where 書式　where memberp文と組み合わせ，オブジェクト中の指定の要素が何番目に位置しているかを番号で返す．	W&L
	・window 書式　window タートルが自由に画面の外に出られるような状態に設定する．	GRAP
	・word 書式　word 〈ワード〉〈ワード〉…… 与えられたワードを合成し，ひとつのワードとして返す．引数が２つ以外のときは全体を（　）でくくる．	W&L

W	・wordp 書式　wordp 〈オブジェクト〉 与えられたオブジェクトがワードであればTRUE，そうでなければFALSEを返す．	W&L
	・wrap 書式　wrap タートルが画面からはずれると反対側から画面に入ってくる状態に設定する．	GRAP
X	・xcor 書式　xcor タートルがいる位置のX座標を返す．	TURTL
Y	・ycor 書式　ycor タートルがいる位置のY座標を返す．	TURTL
記号	・＋ 書式　＋ 〈数値〉〈数値〉…… 与えられた数値の和を返す．引数が２つ以外のときは(　)でくくる．	CALC
	・－ 書式　－ 〈数値〉〈数値〉 与えられた数値の差を返す．	CALC
	・＊ 書式　＊ 〈数値〉〈数値〉…… 与えられた数値の積を返す．引数が２つ以外のときは(　)でくくる．	CALC
	・／ 書式　／ 〈被除数〉〈除数〉 与えられた数値の商を返す．	CALC
	・< 書式　< 〈ワード１〉〈ワード２〉 ワード１が２よりも小さければTRUEを返し，それ以外はFALSEを返す．	IF
	・> 書式　> 〈ワード１〉〈ワード２〉 ワード１がワード２よりも大きければTRUE，そうでなければFALSEを返す．	IF

2　クィック・コマンドリファレンス

6章7節で解説した，N_{88}-BASICから777 BASICへの移植に便利な，クィック・コマンドリファレンスを載せておきます．この表は，左側にN_{88}-BASICのコマンドを網羅してあり，右側にはそれに対応した777 BASICのコマンドが様々な場合を考慮して載せてあります．ですから読者の方は左右の対応を見て，「あっそうか！」と言ってプログラムを書き直すなり，777 BASICのマニュアルを引き直すなりすれば，次第に双方のBASICの関係がわかってくるでしょう．なお，中央の部分に〝like〞とか〝use〞とか書かれている場合について説明します．

LIKE…………・完全に同一仕様ではない
・意味的には同様のものである
・一番似ているものである
・マニュアルを見る必要がる

USE……………・これを使えば，似たようなことができる
・組み合わせて使う
・本質的には違うが，これを使う
・マニュアルを見る必要がある

何も書いていないものは，ほとんどそのまま使えますが，書式を間違わないようにしてください．なお777 BASICの欄に“／”が入っているものは，対応する命令がない，または実現が非常に難しいものです．

さらに，777 BASICの命令に書き換えたつもりだがエラーが出るときには，右側から探してN_{88}-BASICの命令を確認した上で，マニュアルを調べてみることをお勧めします．

このリファレンスを見るより，マニュアルをひいた方が速いと感じるようになれば，かなり移植に慣れることができたと言えます．

N_{88}-BASICと777 BASICは，かなり似た構成をとっているので覚えるのも速いと思います．他のBASICにも，このようなリファレンスを考えてみてはいかがでしょうか．

表A-2 N_{88}-BASICからのクイック・コマンドリファレンス

N88		SMC777	N88		SMC777
ABS		ABS	ERL		ERL
ASC		ASC	ERR		ERR
ATN		ATN	ERROR		ERROR
AUTO		AUTO	EXP		EXP
BEEP	like	BEEP	FIELD		/
BLOAD	use	LINK	FILES	like	DIR
BSAVE		/	FIX		FIX
CALL	like	CALL	FN		FN
CDBL	like	CFLT	FOR~NEXT		FOR~NEXT
CHAIN	like	CHAIN	FPOS		/
CHR$		CHR$	FRE	like	FRE
CINT		CINT	GET		/
CIRCLE	like	CIRCLE	GET@	like	GGET
CLEAR	use	CLEAR, MCLEAR	GOSUB~RETURN		GOSUB~RETURN
CLOSE		CLOSE	GOTO		GOTO
CLS	like	CCLEAR	HELP ON/OFF/STOP		/
CLS 2	like	GCLEAR	HEX$		HEX$
CLS 3	like	WIPE	IF~THEN~ELSE		IF~THEN~ELES
COLOR	use	CCOLOR, GCOLOR	INKEY$, INPUT$	like	INKEY$
COLOR=(,)	use	DEF PALETTE, PALETTE RESET	INP		INP(use carefully)
COM ON/OFF/STOP		/	INPUT	like	INPUT
COMMON		/	INPUT #	like	INPUT #
CONSOLE	like	CONSOLE	INPUT WAIT		/
CONT		CONT	INSTR		INSTR
COPY	like	_COPY	INT		INT
COS		COS	KEY		DEF KEY
CSNG	like	CFLT	KEY LIST		KEY LIST
CSRLIN		CSRLIN	KEY(n) ON/OFF/STOP		/
CVI, CVS, CVD		/	KILL		ERA
DATA		DATA	LEFT$		LEFT$
DATE$		/	LEN		LEN
DEF FN		DEF FN	LET		LET
DEF USR	like	DEF ANN(use carefully)	LFILES		/
DEFDBL	like	DEF FLT	LINE	use	LINE, BOXE, BOXF
DEFINT		DEF INT	LINE INPUT	like	INPUT/L
DEFSNG	like	DEF FLT	LINE INPUT #	like	INPUT/L #
DEFSTR		DEF STR	LINE INPUT WAIT		/
DELETE		DELETE	LIST		LIST
DIM		DIM	LLIST		LLIST
DSKF		/	LOAD	like	LOAD
DSKI$		/	LOAD?		LOAD/V
DSKO$		/	LOC		/
EDIT	like	EDIT	LOCATE	like	LOCATE
END		END	LOCATE,,0		CURSOR OFF
EOF		EOF	LOCATE,,1		CURSOR ON
ERASE		/	LOF		/

N88		SMC777	N88		SMC777
LOG		LN	PUT@	like	GPUT, KCOLOR, KPRINT
LPOS		/	RANDOMIZE		RANDOMIZE
LPRINT		PRINT@ 1	READ		READ
LPRINT USING	like	PRINT@ 1, USING	REM		REM
LSET		/	RENUM		RENUM
MAP		/	RESTORE		RESTORE
MERGE		MERGE	RESUME		RESUME
MID$	use	MID$, REPLACE$	RIGHT$		RIGHT$
MKI$, MKS$, MKD$		/	RND		RND
MON		/	ROLL		/
MOTOR		MOTOR	RSET		/
NAME~AS~		REN~TO~	RUN		RUN
NEW		NEW	SAVE	like	SAVE
NEW ON		/	SCREEN	like	GMODE
OCT$		/	SEARCH		/
ON COM GOSUB		/	SET		/
ON ERROR GOTO	use	ON ERROR GOTO, ERROR ON	SGN		SGN
ON ERROR GOTO 0		ERROR OFF	SIN		SIN
ON HELP GOSUB		/	SPACE$, SPC		SPACE$
ON KEY GOSUB		/	SQR		SQR
ON PEN GOSUB		/	STOP		STOP
ON STOP GOSUB		/	STOP OFF	like	ESC ON
ON TIME$ GOSUB		/	STOP ON	like	ESC OFF
ON~GOSUB		ON~GOSUB	STR$	like	STR$
ON~GOTO		ON~GOTO	STRING$	like	STRING$
OPEN(for input)		OPEN/I	SWAP		SWAP
OPEN(for output)		OPEN/O	TAB		TAB
OPEN(for random access)		/	TAN		TAN
OPTION BASE		OPTION BASE	TERM		/
OUT	like	OUT(use carefully)	TIME$		/
PAINT		PAINT	TIME$ ON/OFF/STOP		/
PEEK		PEEK(use carefully)	TROFF	use	DEBUG
PEN		/	TRON	use	DEBUG
PEN ON/OFF/STOP	use	/	USR	like	ANN(use carefully)
POINT		GPOINT, GPOSX, GPOSY	VAL	like	VAL
POKE		POKE(use carefully)	VARPTR	like	VARPTR
POS	like	CSRCLM	VIEW		/
PRESET	like	GPLOT	WAIT		/
PRINT	like	PRINT	WHILE~WEND		WHILE~WEND
PRINT #	like	PRINT #	WIDTH	use	CONSOLE
PRINT #USING	like	PRINT #, USING	WIDTH LPRINT		/
PRINT USING	like	PRINT USING	WINDOW		/
PSET		GPLOT	WRITE		/
PUT		/	WRITE#		/

3 Assembler

(1) MORE

システムメニューにあるTYPEで,アスキー形式のファイルを一応見ることはできるのですが,若干不満もあります.それは,

・画面が横40桁で,特にアセンブラのリスティングファイルは見苦しくなる

・[CTRL]+[S]を押さないと勝手にどんどん表示していってしまい,見たい部分を逃してしまうことがある

などです.そこで,こういった不満を解消するため,「MORE」というSony Filerのコマンドを作ってみました.これは同名のUNIX (4.1BSD)のコマンド(正確にはフィルタと呼ばれなければなりませんが)の主な機能を真似たもので,

MORE 〈ファイル名〉

という形で実行し,画面1ページ分表示すると,

--MORE--

と表示して止まり,このときスペースを押すとさらに1ページ表示します.またスペースの代わりに[↵]を押すと1行表示して止まります.スペースの代わりにDを押すと,11行分表示して止まります.1ページなり11行なりを表示し終らないうちに止めたいときは,[CTRL]+[S]を押し,[CTRL]+[Q]を押すとそこから1ページ分表示してから止まります.[CTRL]+[S]で止まっているときも[↵]は同様の働きをします.そして[ESC]を押せばいつでもシステムメニューに戻ります.

リストA-2にこのMOREのリスティングファイルを示します.

List A-2

```
                      10 ';=============================================
                      20 ';                        More
                      30 ';=============================================
                      40 ';
      00F0 41314101   50 '              org     0f0h
      00F4 0F00F000
      00F8 58000001   60 '              entry   x
      00FC FF00
      00FE 0000       70 '              start
                      80 ';
           0005       90 'bdos          equ     0005h
           005C      100 'fcb           equ     005ch
```

```
                      110 ';
                      120 ';-------------------------------------------------
                      130 ';                         open file
                      140 ';-------------------------------------------------
                      150 ';
0100 115C00           160 'x:       de=fcb
0103 0E0F             170 '         c=15      ;file open
0105 CD0500           180 '         ]bdos
0108 3C               190 '         a+
0109 2021             200 '         ->mocont1,~0
                      210 ';
                      220 ';--- file not found ---
                      230 ';
010B 2E0B             240 '         l=11      ;xcor
010D 260C             250 '         h=12      ;ycor
010F 1600             260 '         d=0
0111 0E15             270 '         c=21      ;move cursor
0113 CF               280 '         ]'08
0114 111B01           290 '         de=momessa
0117 0E09             300 '         c=9       ;CRT-out message
0119 CF               310 '         ]'08
011A C9               320 '         [
                      330 ';
011B 46696C65         340 'momessa: db        'File not found!!',0
011F 206E6F74
0123 20666F75
0127 6E642121
012B 00
                      350 ';
                      360 ';-------------------------------------------------
                      370 ';                            main
                      380 ';-------------------------------------------------
                      390 ';
012C 0E15             400 'mocont1: c=21
012E 11001B           410 '         de=1b00h          ;console 80
0131 CF               420 '         ]'08
                      430 ';
0132 ED736401         440 '         (spstack)=sp
0136 111850           450 '         de=5018h          ;50h=80,17h=24
                      460 ';
0139 CDEC01           470 'moloop:  ]readdma
013C CD1102           480 '         ]scrnout
013F FE0D             490 '         a-13
0141 2004             500 '         ->monotcr,~0
0143 1650             510 '         d=80
0145 1813             520 '         ->moinkey
                      530 ';
0147 FE0A             540 'monotcr: a-10
0149 2006             550 '         ->monotlf,~0
014B 1D               560 '         e-
014C CC6B01           570 '         ]prntmore,0
014F 1809             580 '         ->moinkey
                      590 ';
0151 15               600 'monotlf: d-
0152 2006             610 '         ->moinkey,~0
0154 1D               620 '         e-
0155 CC6B01           630 '         ]prntmore,0
0158 1800             640 '         ->moinkey
                      650 ';
015A CD1A02           660 'moinkey: ]inkey0
015D FE13             670 '         a-19      ;=^s
015F CCCA01           680 '         ]ctrls,0
0162 18D5             690 '         ->moloop
                      700 ';
0164 0000             710 'spstack: ds        2
                      720 ';
                      730 ';-------------------------------------------------
                      740 ';                         Abort
                      750 ';-------------------------------------------------
                      760 ';
0166 ED7B6401         770 'abort:   sp=(spstack)
016A C9               780 '         [
```

```
                 790 ';
                 800 ';--------------------------------------------------
                 810 ';   Print '--More--' and wait for a space
                 820 ';--------------------------------------------------
                 830 ';
016B 110017      840 'prntmore:de=1700h         ;cursor off
016E 0E15        850 '         c=21
0170 CF          860 '         J'08
                 870 ';
0171 113705      880 'pmprmore:de=0537h         ;set attr to 'reverse'
0174 0E15        890 '         c=21
0176 CF          900 '         J'08
                 910 ';
0177 11B601      920 '         de=pmmessa
017A 0E09        930 '         c=9              ;massage out
017C CF          940 '         J'08
                 950 ';
017D 0E15        960 '         c=21
017F 111705      970 '         de=0517h         ;make attr normal
0182 CF          980 '         J'08
                 990 ';
0183 CD2A02     1000 'pmloop:  Jinkey1
0186 FE0D       1010 '         a-13
0188 200B       1020 '         ->pmnotcr,~0
018A 11C001     1030 '         de=pmmessa1
018D 0E09       1040 '         c=9              ;message out
018F CF         1050 '         J'08
0190 CDDE01     1060 '         Joneline
0193 18DC       1070 '         ->pmprmore
                1080 ';
0195 FE20       1090 'pmnotcr: a-32
0197 2004       1100 '         ->pmnotspc,~0
0199 1E17       1110 '         e=23
019B 1808       1120 '         ->pmreturn
                1130 ';
019D E6DF       1140 'pmnotspc:a=a&0dfh
019F FE44       1150 '         a-44h            ;='D'
01A1 20E0       1160 '         ->pmloop,~0
01A3 1E0B       1170 '         e=11
                1180 ';
                1190 ';---- return from '--More--' ----
                1200 ';
01A5 D5         1210 'pmreturn:#=de
01A6 11C001     1220 '         de=pmmessa1
01A9 0E09       1230 '         c=9              ;message out
01AB CF         1240 '         J'08
                1250 ';
01AC 0E15       1260 '         c=21
01AE 110117     1270 '         de=1701h         ;cursor on
01B1 CF         1280 '         J'08
                1290 ';
01B2 D1         1300 '         de=#
01B3 1650       1310 '         d=80
01B5 C9         1320 '         [
                1330 ';
01B6 2D2D4D6F   1340 'pmmessa: db        '--More--',13,0
01BA 72652D2D
01BE 0D00
01C0 20202020   1350 'pmmessa1:db        '        ',13,0
01C4 20202020
01C8 0D00
                1360 ';
                1370 ';--------------------------------------------------
                1380 ';              Transaction for ctrl S
                1390 ';--------------------------------------------------
                1400 ';
01CA CD2A02     1410 'ctrls:   Jinkey1
01CD FE0D       1420 '         a-13
01CF 2005       1430 '         ->csnotcr,~0
01D1 CDDE01     1440 '         Joneline
01D4 18F4       1450 '         ->ctrls
```

```
                 1460 ';
01D6 FE11        1470 'csnotcr: a-17            ;=^q
01D8 20F0        1480 '         ->ctrls,~0
01DA 111750      1490 '         de=5017h        ;50h=80,18h=23
01DD C9          1500 '         [
                 1510 ';
                 1520 ';---------------------------------------------------
                 1530 ';                 Screen out till lf
                 1540 ';---------------------------------------------------
                 1550 ';
01DE CDEC01      1560 'oneline: ]readdma
01E1 CD1102      1570 '         ]scrnout
01E4 FE0A        1580 '         a-10            ;=lf
01E6 C8          1590 '         [,0
01E7 CD1A02      1600 '         ]inkey0
01EA 18F2        1610 '         ->oneline
                 1620 ';
                 1630 ';---------------------------------------------------
                 1640 ';              read 1 byte from dma area
                 1650 ';---------------------------------------------------
                 1660 ';
01EC 3A1002      1670 'readdma: a=(rdcntr)
01EF B7          1680 '         a=a|a
01F0 2010        1690 '         ->rdskip,~0
                 1700 ';
                 1710 ';---- read one sector ----
                 1720 ';
01F2 D5          1730 '         #=de
01F3 115C00      1740 '         de=fcb
01F6 0E14        1750 '         c=20            ;read one sector
01F8 CD0500      1760 '         ]bdos
01FB 3C          1770 '         a+
01FC CA6601      1780 '         =>abort,0
01FF D1          1790 '         de=#
0200 3E80        1800 '         a=80h
                 1810 ';
0202 2600        1820 'rdskip:  h=0
0204 6F          1830 '         l=a
0205 3C          1840 '         a+
0206 321002      1850 '         (rdcntr)=a
0209 7E          1860 '         a=(hl)
020A FE1A        1870 '         a-1ah           ;=eof
020C CA6601      1880 '         =>abort,0
020F C9          1890 '         [
                 1900 ';
0210 00          1910 'rdcntr:  db      0
                 1920 ';
                 1930 ';---------------------------------------------------
                 1940 ';              Screen out 1 charactor
                 1950 ';---------------------------------------------------
                 1960 ';
0211 D5          1970 'scrnout: #=de
0212 F5          1980 '         #=af
0213 5F          1990 '         e=a
0214 0E08        2000 '         c=8
0216 CF          2010 '         ]'08
0217 F1          2020 '         af=#
0218 D1          2030 '         de=#
0219 C9          2040 '         [
                 2050 ';
                 2060 ';---------------------------------------------------
                 2070 ';                      inkey$(0)
                 2080 ';---------------------------------------------------
                 2090 ';
021A DB1B        2100 'inkey0:  a=p(1bh)        ;key status
021C 0F          2110 '         >(.)@
021D 3803        2120 '         ->i0pressd,@
021F 3E00        2130 '         a=0
0221 C9          2140 '         [
```

```
0222 DB1A       2150 ´i0pressd:a=p(1ah)          ;key data
0224 FE1B       2160 ´          a-27
0226 C0         2170 ´          [,~0
0227 C36601     2180 ´          =>abort
                2190 ´;
                2200 ´;-----------------------------------------------
                2210 ´;                          inkey$(1)
                2220 ´;-----------------------------------------------
                2230 ´;
022A DB1B       2240 ´inkey1:   a=p(1bh)          ;key status
022C 0F         2250 ´          >(.)@
022D 30FB       2260 ´          ->inkey1,~@
022F 18F1       2270 ´          ->i0pressd
                2280 ´;
0231            2290 ´          end

0166 ABORT      0005 BDOS      01D6 CSNOTCR    01CA CTRLS     005C FCB
0222 I0PRESSD   021A INKEY0    022A INKEY1     012C MOCONT1   015A MOINKEY
0139 MOLOOP     011B MOMESSA   0147 MONOTCR    0151 MONOTLF   01DE ONELINE
0183 PMLOOP     01B6 PMMESSA   01C0 PMMESSA1   0195 PMNOTCR   019D PMNOTSPC
0171 PMPRMORE   01A5 PMRETURN  016B PRNTMORE   0210 RDCNTR    0202 RDSKIP
01EC READDMA    0211 SCRNOUT   0164 SPSTACK    0100 X
```

(2) ザイログニーモニックからANN表記への対応表

Debuggerのマニュアルに ANN表記からザイログ表記への対応表は載っていますが，その逆がありません．むしろ必要なのは逆のほうではないかと思うのですが…．そこで，ザイログ表記からANN表記への対応表を表A－3に示します．

j，n n，q q，r，x，yは次のようなものを代表するものです．

- j　　：1文字
- n n　：アドレス
- q q　：AFも含めたレジスタペア
- r　　：(HL)や(IX＋d)等も含めて8ビットのレジスタとイミディエイトのデータ
- x，y：，（カンマ）を含まないもの

表A-3　ザイログ表記からANN表記への対応表

ザイログ表記	ANN表記	ザイログ表記	ANN表記
ADC　x, y	x=x+y@	JR　NC, nn	−>nn, ~@
ADD　x, y	x=x+y	JR　NZ, nn	−>nn, ~0
AND　r	a=a&r	JR　Z, nn	−>nn, 0
BIT　j, r	r. j?	JR　nn	−>nn
CALL　C, nn	]nn, @	LD　x, y	x=y
CALL　M, nn	]nn, −	LDD	(de−)=(hl−)
CALL　NC, nn	]nn, ~@	LDDR	{(de−)=(hl−)}
CALL　NZ, nn	]nn, ~0	LDI	(de+)=(hl+)
CALL　P, nn	]nn, +	LDIR	{(de+)=(hl+)}
CALL　PE, nn	]nn, e	NEG	−a
CALL　PO, nn	]nn, o	NOP	nop
CALL　Z, nn	]nn, 0	OR　r	a=a¦r
CALL　nn	]nn	OTDR	{p(c)=(hl−)}
CCF	~@	OTIR	{p(c)=(hl+)}
CP　r	a−r	OUT　(C), r	p(c)=r
CPD	a−(hl−)	OUT　(n), A	p(n)=a
CPDR	{a−(hl−)}	OUTD	p(c)=(hl−)
CPI	a−(hl+)	OUTI	p(c)=(hl+)
CPIR	{a−(hl+)}	POP　qq	qq=#
CPL	~a	PUSH　qq	#=qq
DAA	bcd	RES　j, r	r. j=0
DEC　x	x−	RET	[
DI	di	RET　C	[, @
DJNZ　nn	~>nn	RET　M	[, −
EI	ei	RET　NC	[, ~@
EX　(SP), qq	(sp) : qq	RET　NZ	[, ~0
EX　AF, AF'	af : af'	RET　P	[, +
EX　DE, HL	de : hl	RET　PE	[, e
EXX	* : *'	RET　PO	[, o
HALT	hlt	RET　Z	[, 0
IM　0	im0	RETI	[i
IM　1	im1	RETN	[n
IM　2	im2	RL　r	<{@r}
IN　A, (n)	a=p(n)	RLA	<{@.}
IN　r, (C)	r=p(c)	RLC　r	<@{r}
INC　x	x+	RLCA	<@{.}
IND	(hl−)=p(c)	RLD	a<<(hl)
INDR	{(hl−)=p(c)}	RR　r	>{r@}
INI	(hl+)=p(c)	RRA	>{.@}
INIR	{(hl+)=p(c)}	RRC　r	>{r}@
JP　(qq)	=>(qq)	RRCA	>{.}@
JP　C, nn	=>nn, @	RRD	a>>(hl)
JP　M, nn	=>nn, −	RST　jjH	]'jj
JP　NC, nn	=>nn, ~@	SBC　x, y	x=x−y@
JP　NZ, nn	=>nn, ~0	SCF	@=1
JP　P, nn	=>nn, +	SET　j, r	r. j=1
JP　PE, nn	=>nn, e	SLA　r	<r0
JP　PO, nn	=>nn, o	SRA　r	>{}r@
JP　Z, nn	=>nn, 0	SRL　r	>0r@
JP　nn	=>nn	SUB　r	a=a−r
JR　C, nn	−>nn, @	XOR　r	a=a^r

4　各種コード表

表A－4はキーボード・エンコーダが返すコードの表です．このうちファンクションキーと[H]に関しては，対応するキーコードを変更することができるので，いつでもこうなっているとは限りません．また，このコードとは別に，[SHIFT]や[CTRL]が押されているか否かを知ることができ，BASICのINKEY$により返されるコードは表A－5のようになります．表A－4は直接キーボード・エンコーダを読むときに必要なもので，BASICを普通に使っている限りは表A－5だけで十分です．

表A-4　キーボード・エンコーダが返すコード表

下位＼上位	上位4ビット															
下位4ビット	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		^F2	SPACE	0	@	P	'	p				ー	タ	ミ		
1	F1	DEL	!	1	A	Q	a	q			。	ア	チ	ム		
2	F2	sF3	"	2	B	R	b	r			「	イ	ツ	メ		
3	sF5	^F3	#	3	C	S	c	s			」	ウ	テ	モ		
4	F3	HOME	$	4	D	T	d	t			、	エ	ト	ヤ		
5	sF4	sF1	%	5	E	U	e	u			・	オ	ナ	ユ		
6	F4	←	&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
7	^F4	↑	'	7	G	W	g	w			ァ	キ	ヌ	ラ		
8	BS	sF2	(	8	H	X	h	x			ィ	ク	ネ	リ		
9	TAB	→	)	9	I	Y	i	y			ゥ	ケ	ノ	ル		
A	LF	^F1	*	:	J	Z	j	z			ェ	コ	ハ	レ		
B	F5	ESC	+	;	K	[	k	¦			ォ	サ	ヒ	ロ		
C	^F5	↓	,	<	L	¥	l	¦			ャ	シ	フ	ワ		
D	RETURN	[H]	−	=	M	]	m	¦			ュ	ス	ヘ	ン		
E	CLR		.	>	N	^	n	~			ョ	セ	ホ	゛		
F	INS		／	?	O	_	o	RUB OUT			ッ	ソ	マ	゜		

s……SHIFTキーとの同時押し
^……CTRLキーとの同時押し
F1～F5……ファンクションキー

表A-5　BASICのINKEY$により返されるコード表

	下位＼上位	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	0		^F2	SPACE	0	@	P	'	p	s[illegible]	sINS		ー	タ	ミ		
	1		^DEL	!	1	A	Q	a	q	F1	DEL	。	ア	チ	ム		
	2			"	2	B	R	b	r	F2	sF3	「	イ	ツ	メ		
	3		^F3	#	3	C	S	c	s	sF5	sDEL	」	ウ	テ	モ		
	4		^HOME	$	4	D	T	d	t	F3	HOME	、	エ	ト	ヤ		
下	5			%	5	E	U	e	u	sF4	sF1	・	オ	ナ	ユ		
位	6		^←	&	6	F	V	f	v	F4	←	ヲ	カ	ニ	ヨ		
4	7	^F4	^↑	'	7	G	W	g	w	s←	↑	ァ	キ	ヌ	ラ		
ビ	8	BS		(	8	H	X	h	x	s↑	sF2	ィ	ク	ネ	リ		
ッ	9	TAB	^→	)	9	I	Y	i	y	s→	→	ゥ	ケ	ノ	ル		
ト	A	LF	^F1	*	:	J	Z	j	z	s↓		ェ	コ	ハ	レ		
	B		ESC	+	;	K	[	k	{	F5		ォ	サ	ヒ	ロ		
	C	^F5	^↓	,	<	L	¥	l	¦	sHOME	↓	ャ	シ	フ	ワ		
	D	RETURN	^[illegible]	−	=	M	]	m	}	sCLR	[illegible]	ュ	ス	ヘ	ン		
	E	^CLR		.	>	N	^	n	~	CLR		ョ	セ	ホ	゛		
	F	^INS		/	?	O	_	o	RUB OUT	INS		ッ	ソ	マ	゜		

(上位4ビット)

s ……SHIFTキーとの同時押し
^……CTRLキーとの同時押し
F1〜F5……ファンクションキー

文字フォント表

BASICでFONT RESET 0または1を実行したあとのキャラクタフォントを以下に示します．キャラクタコード0〜31の文字は普通の文字と違って，PRINT等で表示しようとしても，この表にあるような形の「文字」は表示されません．これらの文字は表示を「制御」する文字だからです．たとえばCHR$(10)をPRINTするとカーソルが1段下がるといった具合です．ですからBASICでキャラクタコード0〜31の文字をこの表にあるような形の文字として表示するには，VOUTかCPLOTを使うしかありません．これらの命令の使い方はマニュアルを参照してください．

FONT RESET 1の場合，FONT RESET 0との違いはキャラクタコード128〜159，224〜255のみで，その差分だけ載せてあります．

表A-6 FONT RESET 0の場合の文字フォント

16進コード	00~1F		20~3F		40~5F		60~7F		80~09F		A0~BF		C0~DF		E0~FF	
	コード	文字	コード	文字	コード	文字	コード	文字	コード	文字	コード	文字	コード	文字	コード	文字
0	0		32	スペース	64	@	96	'	128	▁	160		192	タ	224	═
1	1	↑	33	!	65	A	97	a	129	▂	161	。	193	チ	225	╞
2	2	↓	34	"	66	B	98	b	130	▃	162	「	194	ツ	226	╪
3	3	→	35	#	67	C	99	c	131	▄	163	」	195	テ	227	╡
4	4	←	36	$	68	D	100	d	132	▅	164	、	196	ト	228	◢
5	5	♀	37	%	69	E	101	e	133	▆	165	・	197	ナ	229	◣
6	6	♂	38	&	70	F	102	f	134	▇	166	ヲ	198	ニ	230	◥
7	7	♪	39	'	71	G	103	g	135	█	167	ァ	199	ヌ	231	◤
8	8	⧗	40	(	72	H	104	h	136	▏	168	ィ	200	ネ	232	♠
9	9		41	)	73	I	105	i	137	▎	169	ゥ	201	ノ	233	♥
A	10	↯	42	*	74	J	106	j	138	▍	170	ェ	202	ハ	234	♦
B	11	↯	43	+	75	K	107	k	139	▌	171	ォ	203	ヒ	235	♣
C	12	↴	44	,	76	L	108	l	140	▋	172	ャ	204	フ	236	●
D	13	↵	45	−	77	M	109	m	141	▊	173	ュ	205	ヘ	237	○
E	14	☻	46	.	78	N	110	n	142	▉	174	ョ	206	ホ	238	╱
F	15	☺	47	/	79	O	111	o	143	┼	175	ッ	207	マ	239	╲
0	16	⬮	48	0	80	P	112	p	144	┴	176	ー	208	ミ	240	╳
1	17	⬮	49	1	81	Q	113	q	145	┬	177	ア	209	ム	241	円
2	18	⬯	50	2	82	R	114	r	146	┤	178	イ	210	メ	242	年
3	19	▒	51	3	83	S	115	s	147	├	179	ウ	211	モ	243	月
4	20	▯	52	4	84	T	116	t	148	▔	180	エ	212	ヤ	244	日
5	21	▯	53	5	85	U	117	u	149	─	181	オ	213	ユ	245	時
6	22	▓	54	6	86	V	118	v	150	│	182	カ	214	ヨ	246	分
7	23	▯	55	7	87	W	119	w	151	▕	183	キ	215	ラ	247	秒
8	24	⏏	56	8	88	X	120	x	152	┌	184	ク	216	リ	248	▶
9	25	▯	57	9	89	Y	121	y	153	┐	185	ケ	217	ル	249	◀
A	26	▮	58	:	90	Z	122	z	154	└	186	コ	218	レ	250	©
B	27	▯	59	;	91	[	123	¦	155	┘	187	サ	219	ロ	251	®
C	28	▯	60	<	92	¥	124	¦	156	╭	188	シ	220	ワ	252	▯
D	29	▯	61	=	93	]	125	¦	157	╮	189	ス	221	ン	253	☛
E	30	웃	62	>	94	^	126	~	158	╰	190	セ	222	゛	254	〒
F	31	웃	63	?	95	_	127		159	╯	191	ソ	223	゜	255	☎

表A-7　FONT RESET 1の場合の文字フォント(0との差分)

16進コード	80～09F				E0～FF			
	コード	文字	コード	文字	コード	文字	コード	文字
0	128		144	ー	224	た	240	み
1	129	。	145	あ	225	ち	241	む
2	130	「	146	い	226	つ	242	め
3	131	」	147	う	227	て	243	も
4	132	、	148	え	228	と	244	や
5	133	・	149	お	229	な	245	ゆ
6	134	を	150	か	230	に	246	よ
7	135	ぁ	151	き	231	ぬ	247	ら
8	136	ぃ	152	く	232	ね	248	り
9	137	ぅ	153	け	233	の	249	る
A	138	ぇ	154	こ	234	は	250	れ
B	139	ぉ	155	さ	235	ひ	251	ろ
C	140	ゃ	156	し	236	ふ	252	わ
D	141	ゅ	157	す	237	へ	253	ん
E	142	ょ	158	せ	238	ほ	254	゛
F	143	っ	159	そ	239	ま	255	゜

参考文献

(1) 777 Assembler 使用説明書
(2) 777 Debugger 使用説明書
(3) Micro Computer ハードウェア解説書
(4) Micro Computer 取扱説明書
(5) 777 BASIC ハンドブック
(6) Dr. Logo™ ハンドブック
(7) Dr. Logo™ テキストブック
(8) 777 Memo 使い方
(9) Micro Computer 回路図集
　　— 以上ソニー株式会社発行 —
(10) PC-8801グラフィックスのすべて
　　工藤丈彦・榊正憲・田村明史・横溝和宏共著
　　アスキー出版局発行
(11) THE 3DIMENSIONAL COMPUTER・GRAPHICS
　　山本強著　　CQ出版発行

●本書をより理解するための参考書籍(アスキー出版局発行分)

— BASIC —

・「はじめて読むBASIC」　高玉皓司著

マイクロソフト社系BASICの入門書．BASICを全く知らないという方に一読をお勧めします．

・「PC-8801 BASIC入門」

工藤丈彦・屋敷誠二・横溝和宏共著

本書で777 BASICと対比したN_{88}-BASICの入門書．移植に関する節をより理解しようという方にお勧めします．

— Assembler —

・「はじめて読むマシン語」　村瀬康治著

機械語に関する基礎知識を得ようという方に一読をお勧めします．

— OS —

・「入門 CP／M」　村瀬康治著
・「実習 CP／M」　村瀬康治著
・「応用 CP／M」　村瀬康治著

Sony Filer と多くの互換性を持つCP／M の解説書．OS関係に興味のある方にお勧めします．

SMC-777ガイドブック

1984年4月30日　初版発行
1984年10月15日　第1版第2刷発行
定価2,500円

著　者　更科潤二&ANIMAL HOUSE
発行者　塚本慶一郎
発行所　株式会社アスキー
〒107 港区南青山5－11－5住友南青山ビル5F
振　替　東京4-161144
電　話　03－486－7111（代表）

本書は著作権法上の保護を受けています．本書の一部あるいは全部について（ソフトウェア及びプログラムを含む），株式会社アスキーから文書による許諾を得ずに，いかなる方法においても無断で複写，複製することは禁じられています．

編集担当　土田米一
表紙担当　郷　啓子
印刷　壮光舎印刷株式会社

ISBN4-87148-738-5 C3055 ¥2500E

日本音楽著作権協会（出）許諾第8360107-301号

アスキー出版局

ISBN4-87148-738-5 C3055 ¥2500E

SMJ-S029B

ASCII SYSTEM BANK

SMC編第①巻 内容抜粋

ハードとソフトの主な特徴
Sony Filerの概要
Logo初級講座
タートル・グラフィックス入門
Logoのプログラム構造
グローバル変数とローカル変数
ドラゴン曲線を描く
リスト処理関連のプリミティブ
データ管理プリミティブ
777 BASICの特徴
EDITとDEBUG
効果音とサウンド
BASICによるMEMOの活用
グラフィックスの基礎
カラーパレット・テクニック
24色表示
線画による3-Dグラフィックス
光線追跡法
SMCグラフィックスエディターの利用
N88-BASICからの移植テクニック
ハードウェア構成
応用周辺機器の自作
777 Assemblerとは
DOSコール
5番地と8番地の違い
Sony Filerのコマンドを作る
Logoプリミティブ一覧
N88-BASICからのクイック・コマンドリファレンス
ザイログニーモニックからANN表記への対応表

更科潤二 & ANIMAL HOUSE著

アスキー・システム・バンク(SMC#1)

SMC-777 ガイドブック

定価2,500円
ISBN4-87148-738-5
C3055 ¥2500E